KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Technology

Advanced Applications of Knowledge Representation Technology

Aerts Bram

Supervisors:
Prof. dr. Joost Vennekens
Prof. dr. ir. Toon Goedemé

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Technology (PhD)

August 2020

Advanced Applications of Knowledge Representation Technology

Aerts BRAM

Examination committee:
Prof. dr. ir. Hans Rediers, chair
Prof. dr. Joost Vennekens, supervisor
Prof. dr. ir. Toon Goedemé, supervisor
dr. ir. Wannes Meert
dr. ir. Peter Karsmakers
Prof. dr. ir. Kris Aerts
Herman Dubois
(Saint-Gobain)
Prof. dr. Bart Bogaerts
(Free University of Brussels-VUB)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Technology (PhD)

August 2020

© 2020 KU Leuven – Faculty of Engineering Technology
Uitgegeven in eigen beheer, Aerts Bram, Jan Pieter De Nayerlaan 5, B-2860 Sint-Katelijne-Waver (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Preface

It all started for me during a lab session in which we were solving logic puzzles in some exotic modelling language called IDP. I particularly liked those sessions and was intrigued by the elegant way knowledge could be modelled in this knowledge-based language. At a certain moment, Joost Vennekens approached me and asked whether I would be interested in doing a PhD study on a related topic. I must admit that, initially, I had some doubts. Even though the topic interested me, I never pictured myself doing a PhD. However, after some time of consideration, I decided to start a PhD on the topic of "Advanced Applications of Knowledge Representation Technology".

So here we are, almost six years after I started my PhD. Looking back, it has been a very rewarding experience. It allowed me to grow both personally and professionally. This PhD not only allowed me to conduct interesting research projects, it also introduced me to teaching (something I surprisingly enjoyed), and do some interesting collaborations, with colleagues, other research groups and industrial partners. Moreover, it also gave me the opportunity to explore a part of the world, by attending numerous interesting conferences, but also as part of the research collaboration with Saint-Gobain.

There are a number of people without whom this dissertation would never have been possible. First and foremost, I would like to thank my supervisor Joost Vennekens, for giving me the opportunity to start a PhD. Without your continuous support, the heaps of feedback, critical questions and illuminating insights, this PhD would never have been finished. Also, thank you for proposing such interesting research projects and providing the necessary funds to finish this PhD. It was a privilege to be a part of this research group under your supervision.

This dissertation would not have been possible without the interesting research projects and support from the industrial partners, such as Saint-Gobain, Agidens and OTSA. In particular, I would like to thank Herman Dubois for mentoring the main research project in this dissertation, for confiding me the task of knowledge extractor during multiple workshops all over the world, and of course for his never-ending enthusiasm. In addition, I would like to thank Nicholas Decleyre for the additional support during

this project.

Moreover, I would like to thank the members of my examination committee, Toon Goedemé, Wannes Meert, Kris Aerts, Herman Dubois, Peter Karsmakers, Bart Bogaerts and Hans Rediers for all useful advice and feedback on this dissertation.

Another thank you goes out to my colleagues at the EAVISE research group. Even though I saw quite a number colleagues come and go, the atmosphere in this group of people was always excellent. I really enjoyed the in-depth discussions, the interesting research collaborations, the numerous coffee breaks and the enjoyable extracurricular activities, such as playing pool, karting, kayaking, and paintballing.

I would also like to thank some members from other research groups for the fruitful collaborations, in particular Ingmar Dasseville, Pierre Carbonnelle, Dries Van Daele, Wannes Meert, Jay Chakravarti and the other members of the CAMETRON research project.

To conclude, I would like to say thank you to my family, and in particular, my mother. She always supported me whenever necessary, in any way she could and without any hesitation. Without her I would not be where I am today. And thank you Veerle, for being you and supporting me throughout the last years of this adventure.

Bram Aerts Sint-Katelijne-Waver, May 2020.

Abstract

Knowledge representation is a field of research that focuses on representing real world information in an expressive form, in order to apply reasoning tasks to the represented knowledge to solve complex problems. The Knowledge Base Paradigm states that this domain knowledge can be modelled independently of the inference tasks that are executed on this knowledge. This enables to support a variety of powerful inference tasks on the same knowledge specification. While this field of research is extensively studied in the literature, this principle rarely finds its way to practical applications in industry.

In this dissertation, we investigate the practical usability of knowledge representation technologies in real life industrial applications. To this end, a number of case studies have been selected from different domains, each with different requirements. In these use cases, the practical usability of knowledge representation technologies is demonstrated. These techniques are particularly useful to provide extensive decision support to experts.

The research is motivated by the many shortcomings we identified in existing approaches, whilst executing a broad variety of case studies. First, existing systems typically focus on replicating the tasks of human experts, while little attention is paid to supporting them in those tasks. Second, knowledge is seldom formulated in a readable form. However, this practice facilitates knowledge elicitation and improves transparency and maintainability of the resulting model. Third, even though knowledge representation techniques are extensively studied in the literature, they are rarely used in real world industrial applications.

Throughout this dissertation, we examine how techniques from the field of knowledge representation can be used to cope with these shortcomings. According to the knowledge base paradigm, a vast variety of inference tasks can be executed on the same knowledge base. Drawing on this paradigm, we create a system that enables experts to utilize a number of customized functionalities, offering them a wide-ranging support, all based on the same domain knowledge specification. We propose the expressive,

intuitive and readable cDMN notation, which enables to provide this support based on a knowledge specification that is readable and maintainable by domain experts themselves. Finally, we show that machine learning techniques can be used to analyse historical data to complement expert know-how.

Beknopte samenvatting

Kennisrepresentatie is een onderzoeksdomein dat zich concentreert op het voorstellen van informatie over de echte wereld in een expressieve vorm, met als doel met deze kennis te kunnen redeneren op een automatische manier. Het kennisbankparadigma stelt dat domeinkennis afzonderlijk kan worden gemodelleerd van de inferentietaken die op deze kennis worden uitgevoerd. Dit maakt het mogelijk om uiteenlopende, krachtige inferenties uit te voeren op één enkele kennisspecificatie. Hoewel dit onderzoeksdomein uitgebreid wordt onderzocht in de wetenschappelijke literatuur, wordt het principe zelden gebruikt in praktische industriële context.

In dit doctoraat onderzoeken we de praktische toepasbaarheid van kennisrepresentatietechnieken in industriële applicaties. Met dit doel voor ogen, hebben we een aantal uiteenlopende casestudies geselecteerd uit verschillende domeinen, die elk hun eigen specifieke eisen stellen. De praktische bruikbaarheid van kennisrepresentatietechnologieën wordt aangetoond in deze casestudies. De technieken zijn onder meer bijzonder nuttig bij het aanbieden van een uitgebreide beslissingsondersteuning voor experts.

De motivatie voor het onderzoek zijn de vele tekortkomingen die we tegenkwamen in de uiteenlopende casestudies die we uitvoerden. Ten eerste focussen de bestaande systemen zich doorgaans op het nabootsen van de taken van menselijke experts, terwijl weinig aandacht wordt besteed aan hen te ondersteunen bij deze taken. Ten tweede, alhoewel een leesbare voorstelling van domeinkennis de kennisextractie vergemakkelijkt en de betrokkenheid van de experts bij de voorgestelde kennis bevordert, wordt de leesbaarheid van zulke kennis vaak over het hoofd gezien. Ten derde, desondanks dat er veel onderzoek gebeurt naar kennisrepresentatie, vinden we deze technieken nauwelijks terug in industriële toepassingen.

Doorheen de doctoraatstekst onderzoeken we hoe technieken uit het domein van kennisrepresentatie kunnen worden benut om deze tekortkomingen te verhelpen. Volgens het kennisbankparadigma kan een heel aantal inferentietaken worden toegepast op dezelfde kennisbank. Op basis van dit paradigma, maken we een systeem dat experts in staat stelt om gebruik te maken van verschillende op maat gemaakte functionaliteiten,

volledig gebaseerd op éénzelfde kennisspecificatie. Bovendien introduceren we een expressieve, intuïtieve en leesbare notatie, genaamd cDMN. Deze notatie zorgt ervoor dat experts toegang hebben tot de bovenvermelde functionaliteiten op basis van een kennisspecificatie die leesbaar is en die ze zelf kunnen onderhouden. Tenslotte tonen we aan dat machine learning technieken kunnen worden gebruikt om historische data te analyseren om zo de kennis van experts aan te vullen.

List of Abbreviations

AutoConfig An interactive knowledge-based configuration system.. xiii, xiv, 29, 55–63, 66–69, 78, 79, 136

cDMN Constraint DMN. x, xiii–xv, 9–12, 14–22, 24–28, 32, 33, 70, 71, 73–79, 81, 84, 107–109, 132, 133, 135–137

CP-logic Causal Probabilistic Logic, a PLP language.. 84, 85, 107

DMN Decision Model and Notation. ix, xiii, 9–18, 20–24, 26–28, 31–33, 35–46, 48, 51, 53–56, 70, 73, 74, 76–79, 132, 135

DRD Decision Requirement Diagram. xiii, 11, 36, 38, 40, 42

FEEL Friendly Enough Expression Language, this language can be used to evaluate expressions in DMN.. xiii, 12, 13, 21, 22, 24, 78

FO(·) First Order Logic with extensions, the input language of IDP. x, 9, 17, 18, 20, 28, 32–34, 46, 50, 70, 71, 76, 107, 147

FO First Order Logic. 19, 33, 53, 55, 70

IDP Imperative Declarative Programming. 9, 18, 20, 21, 28, 29, 32–34, 49, 51–56, 58, 60, 63, 71, 78–80, 136

KB Knowledge Base.. 28, 31–33, 58, 60, 62, 80, 136

KR Knowledge Representation. 130

ML Machine Learning. 63–66, 68, 114, 134

PLP Probabilistic Logic Programming. 81, 83, 84

ProbLog An inference system for CP-logic.. 84–86, 88, 90, 98, 99, 101, 106, 107, 109, 111, 113

S-FEEL Simple Friendly Enough Expression Language, a simplified variant of the FEEL language.. 12, 13, 15, 16, 78

Contents

Abstract iii

Beknopte samenvatting v

List of Abbreviations viii

List of Symbols ix

Contents ix

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Knowledge Representation and Artificial Intelligence 1
- 1.2 Research Topics 2
  - 1.2.1 Sources of Knowledge 2
  - 1.2.2 Readability 3
  - 1.2.3 Functionality Requirements 4
  - 1.2.4 Presentation of Results 5
  - 1.2.5 Combination of Knowledge Sources 5
- 1.3 Contributions 6
- 1.4 Outline of Research 7

2 cDMN: A Tight Integration of DMN and Constraint Reasoning 9
- 2.1 Introduction 10
- 2.2 Preliminaries 11
  - 2.2.1 DMN 11
  - 2.2.2 IDP and FO(·) 12
- 2.3 Challenges Overview 14

2.4 Related Work . . . 14
2.5 cDMN: Syntax & Semantics . . . 16
2.5.1 Glossary . . . 16
2.5.2 Decision Tables and Constraint Tables . . . 17
2.5.3 Data Tables . . . 19
2.5.4 Execute Table . . . 20
2.5.5 Semantics of cDMN . . . 20
2.6 Implementation . . . 26
2.7 Results and Discussion . . . 26
2.7.1 Constraint Tables . . . 26
2.7.2 Quantification . . . 27
2.7.3 Optimization . . . 28
2.7.4 Overview of Challenges . . . 29
2.8 Conclusions and Future Work . . . 32

3 Machine Component Design **33**
3.1 Introduction . . . 35
3.2 Preliminaries . . . 38
3.2.1 Decision Model and Notation . . . 38
3.2.2 IDP and FO(·) . . . 38
3.3 Running Example . . . 39
3.4 Modelling the Design Process . . . 40
3.4.1 Knowledge Extraction of the Design Process . . . 40
3.4.2 DMN Model of Running Example . . . 42
3.4.3 Results . . . 46
3.4.4 Discussion . . . 47
3.5 Modelling the Underlying Constraints . . . 48
3.5.1 Knowledge Extraction of the Physical Constraints . . . 49
3.5.2 Knowledge Extraction of the Preferences . . . 50
3.5.3 IDP Model of Running Example . . . 51
3.5.4 Missing Data . . . 56
3.5.5 Implementation of a Constraint-based Approach . . . 57
3.5.6 Results . . . 58
3.5.7 Discussion . . . 59
3.6 Interactive Decision Support . . . 60
3.6.1 Existing Functionalities . . . 61
3.6.2 Additional Functionalities . . . 64
3.6.3 Using Historical Data . . . 67
3.6.4 Evaluation . . . 72
3.7 cDMN: A user-friendly representation of constraint knowledge . . . 74
3.7.1 Glossary . . . 75
3.7.2 Dimensions . . . 75
3.7.3 Design Type and Spacer . . . 76

3.7.4 Materials . . . 77
3.7.5 Shrinkage check . . . 77
3.7.6 Preferences . . . 78
3.7.7 Discussion . . . 81
3.8 Related Work . . . 82
3.9 Conclusions and future work . . . 83

4 A Probabilistic Logic Programming Approach to Automatic Video Montage 85
4.1 Introduction . . . 86
4.2 Preliminaries: CP-logic and the ProbLog System . . . 88
4.3 Cinematographic Model . . . 90
4.4 Overview of the Editing System . . . 92
4.4.1 Data Representation . . . 93
4.4.2 Cinematographic Model . . . 94
4.5 Using the Cinematographic Model . . . 102
4.6 Achieving Real-Time Performance . . . 103
4.7 Discussion of Probabilities . . . 105
4.7.1 Accumulated Probability . . . 105
4.7.2 Quality of Montage . . . 106
4.7.3 Effect of P-frames . . . 108
4.7.4 Discussion . . . 109
4.8 Experimental Results . . . 109
4.9 Readable Representation . . . 110
4.10 Parameter Learning . . . 114
4.11 Related Work . . . 116
4.12 Conclusions and Future Work . . . 117

5 Alarm Management on a Liquid Bulk Terminal 118
5.1 Introduction . . . 119
5.2 Related Work . . . 120
5.3 Project goals . . . 121
5.4 Identifying Interesting Alarms . . . 121
5.5 Automated Analysis of Alarms . . . 125
5.5.1 Dataset . . . 125
5.5.2 Algorithms . . . 127
5.5.3 Visualization . . . 130
5.5.4 Graphical Interface . . . 132
5.6 Results . . . 132
5.7 Conclusions And Future Work . . . 133

6 Conclusions and Future Work 135
6.1 General Conclusions and Lessons Learned . . . 135

6.1.1 Sources of knowledge 136
6.1.2 Readability 138
6.1.3 Functionality 139
6.1.4 Presentation 140
6.1.5 Combining Knowledge Sources 141
6.2 Future Work 141

Bibliography **145**

About the Author **153**

List of Publications **155**

List of Figures

1.1 Graphical representation of the research topics in this dissertation. . . 3

2.1 Visualization of the research topics in this chapter. 10
2.2 Decision table to define whether a person is an adult. 12
2.3 Extract of the map colouring solution in standard DMN with FEEL. . 13
2.4 Example glossary in cDMN. 16
2.5 Implementation of "Nobody hates everyone" in *Who Killed Agatha?*. . 19
2.6 Constraint table with quantification in cDMN 19
2.7 Data table describing countries and their neighbours. 20
2.8 Examples of execute tables. 20
2.9 Decision table that determines the charge of a person 24
2.10 Decision table that counts the number of invited guests. 25
2.11 Extract of *Monkey Business* implementation in OpenRules and cDMN. 27
2.12 Extract of the *Monkey Business* implementation in Corticon and cDMN. 28
2.13 Decision tables and constraint table for Balanced Assignment. 28
2.14 Calculating the food properties of burgers in Corticon and cDMN. . . 30
2.15 Implementation of "A killer always hates and is no richer than their victim" in OpenRules and cDMN 31

3.1 Visualization of the research topics in this chapter. 34
3.2 Two different approaches for standard and non-standard designs. . . . 37
3.3 DRD of the product design application. 43
3.4 Decision table for dimensions. 43
3.5 Decision tables for Design Type and Spacer. 44
3.6 Decision tables to determine component materials. 44
3.7 Decision tables to check whether a design can cope with the expected shrinkage. 45
3.8 Propagation in the AutoConfig tool. 62
3.9 Explanation in the AutoConfig tool, consisting of the user-selected atoms (top) and constraints (bottom). 63

3.10 Comparison between an open and a closed design in the AutoConfig tool. 64
3.11 Visual representation of when a constraint is deactivated in the AutoConfig tool. 66
3.12 Indication of a constraint that cannot be reactivated in the AutoConfig tool 66
3.13 Browsing historical products in the AutoConfig tool. 70
3.14 Extract of the cDMN glossary for the product design application. . . . 75
3.15 Decision table for dimensions. 76
3.16 Constraint tables for the selection of Design Type and Use Component. 76
3.17 Tables for the selection of component materials. 77
3.18 Constraint approach to determine whether the shrinkage load can be handled. 78
3.19 Preferences in cDMN. 80

4.1 Visualization of the research topics in this chapter. 86
4.2 Making a montage out of video streams. 91
4.3 Overview of the editing system. 93
4.4 Probability of a shot being considered too short. 96
4.5 Execution time vs. length of p-frames (in seconds). 104
4.6 Accumulated probability of switching to a camera. 106
4.7 Quality of a montage where a switch is postponed θ frames. 107
4.8 Quality comparison of switching at T_0 and ignoring a trigger for θ frames. 108
4.9 Confidence of subjects in their choice. 111
4.10 Extract of the decision logic of montage knowledge in cDMN. 112
4.11 Extract of the cDMN glossary of the montage knowledge. 113

5.1 Visualization of the research topics in this chapter. 119
5.2 Visualization of alarm occurrences. 122
5.3 Comparison of alarms with a low and high Poisson λ 123
5.4 Proposed graphical interface. 125
5.5 Comparison of the weighted impurity in decision trees trained on unbalanced and balanced data. 129
5.6 Visualization of a decision tree. 131
5.7 GUI for the decision tree learner. 133
5.8 Table representation of an alarm prediction. 134

6.1 Graphical representation of the research topics in this dissertation. . . 137

List of Tables

2.1 Percentage of occurrence of properties in challenges. 14
2.2 Comparison of the number of cells used per implementation. 29
2.3 Comparison between the problem properties and their cDMN answers. 32

4.1 Probabilities of different shot transitions. 99

Chapter 1

Introduction

Today, the demand for assistance of domain experts in their daily jobs keeps increasing. Expert systems offer a solution to some extent, but they have their limitations. Such systems focus on automating the decision-making of a human expert, in order to solve complex problems [36, 2, 1]. Typically, they consist of a collection of facts and rules and an inference engine, which is used to draw logical conclusions. However, these systems are usually designed to replicate the decision-making of experts, not to support them in decision-making. Additionally, the functionalities of such systems are limited. We aim to assist the experts in different ways, by providing a multitude of functionalities.

In this PhD we investigate how the field of Knowledge Representation (KR) can be used to build a system that can assist domain experts in a number of ways. Two types of knowledge can be distinguished: expert know-how and data. We examine whether knowledge from both of these sources can be represented in a readable format. Furthermore, we investigate what functionalities are required to use this knowledge in order to assist experts in their decision-making, and how to present these functionalities in a comprehensible way. Finally, this thesis focuses on how to combine these two sources of knowledge.

1.1 Knowledge Representation and Artificial Intelligence

Knowledge Representation (KR) is a field of Artificial Intelligence (AI) that concentrates on representing real world information in an expressive and practical

form. Two issues are central here. A first focus lies in finding a representation that is expressive enough to represent the necessary knowledge, while assuring readability for humans too. A second focus lies in reasoning with this knowledge, i.e. applying different kinds of reasoning tasks to the represented knowledge to solve complex problems. KR asserts that the conventional programming approach (i.e. coding the procedural process to find a solution) is not suited to solve complex problems. KR allows such complex problems to be modelled more easily, while also improving maintainability. Instead of focusing on *how* to solve a problem (i.e. code a procedure to come to a solution), KR focuses on *what* the problem is (i.e. what does a solution look like).

Besides KR, AI consists of a number of additional fields. One of these fields is Machine Learning (ML). Where KR starts from a manually defined representation of the world, ML techniques use data to construct a predictive model. Contrary to KR techniques, the main interest of ML is the predictive quality of those models, while little attention is given to the representation of knowledge in models. These techniques, however, come in very useful when no expert knowledge is available of a certain (sub-)domain.

In this thesis we investigate the added value of AI techniques in a multitude of domains. Our main focus lies in the field of KR. However, for some (sub-)problems, KR is not sufficient to tackle all obstacles. Therefore, we complement the KR-based solution with a number of ML techniques where needed.

1.2 Research Topics

Throughout this thesis we focus on a number of aspects in particular. The graphical representation in Figure 1.1 structures the research topics in this PhD. This section delineates each of these research topics. In the remainder of this dissertation, we use this graphical structure to illustrate the coherence between the research topics in the different case studies.

1.2.1 Sources of Knowledge

A first question that may arise is: what kind of knowledge is needed to construct a system that is able to assist experts in a meaningful way? There are several types of knowledge that can be distinguished.

First, and most obviously, knowledge may refer to expert know-how. In general, domain experts have an extensive know-how of their domain. This knowledge can be used as a basis to build a system that is able to assist them and other users with a more limited knowledge of the domain, e.g. junior engineers.

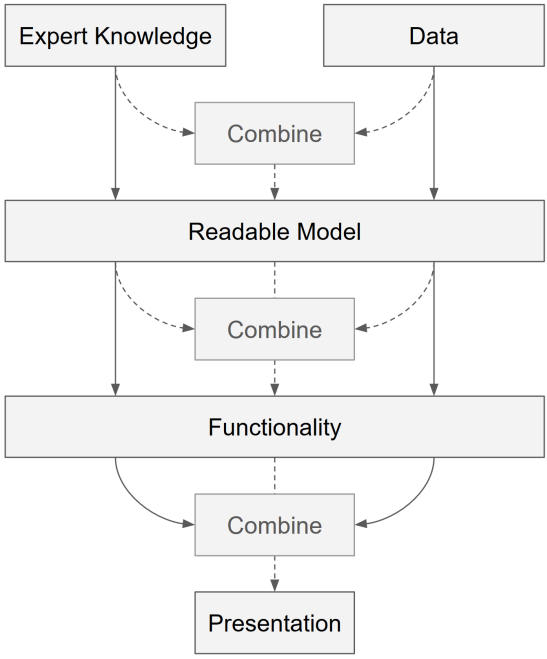


Figure 1.1: Graphical representation of the research topics in this dissertation.

Second, additional knowledge can also be found in data, e.g., logging files that have been captured while monitoring a specific process. This data contains a lot of information about the process at hand, which can be used to build a knowledge model. Moreover, exploring this data can also be helpful for experts to find patterns and implicit knowledge.

1.2.2 Readability

We look into the possibility to represent the knowledge from both of these sources in a readable format.

Expert Know-how

Expert know-how is often used as a basis to build expert systems. The knowledge in such systems can be formulated in numerous languages, e.g., conventional programming languages or KR languages. Domain experts, however, rarely have

an understanding of such languages. Therefore, to build such a system, they rely on people from the IT department, who have limited expertise of the domain at hand. Nevertheless, the full knowledge of the domain experts should be captured in the system. Typically, the procedure to capture all this knowledge involves a lot of communication between domain experts and IT . It is not uncommon that misunderstandings occur. Those lead to erroneous knowledge in the system. Identifying such errors is a difficult process, since the domain experts lack the ability to read the constructed code.

What if we could represent this expert knowledge in such way that domain experts can understand the information represented in the model? This would eliminate the prominent role of the IT department in the process. There are a number of advantages. First, knowledge extraction is made easier. The readability of such models enables the domain experts to build and maintain those models themselves. Second, the transparency of such a system is much higher. Both experts and apprentices can interpret the knowledge in an existing system. Consequently, it allows apprentices to easily gain knowledge of the domain. Moreover, experts can verify the correctness of the knowledge in the system. Third, as a result, the maintainability of such models increases. Last, in addition to the more technical advantages, the involvement of the experts in the creation of such a system also improves their confidence in the resulting system.

Data

ML algorithms excel in using data to build a model that is able to predict the output of a certain task. To do this, such algorithms find patterns in the data and build a model based on these patterns. Such a model contains a lot of information about the domain. Presenting this model in a readable format enables the expert to understand the reasoning of the ML model, and gain knowledge about the domain.

There are two factors that complicate the use of this learned knowledge in a readable model. A first complication is that this knowledge is not extracted from expert know-how, so it is more challenging to find a representation of this knowledge they can understand. Second, ML techniques are designed to construct a model that can predict as accurately as possible, often with little to no attention for interpretability of the model. This conflicts with our goal to use such a ML model to complement knowledge in a readable model.

1.2.3 Functionality Requirements

We aim to build a system that is usable by an end-user. When constructing a knowledge model, we need to make sure this knowledge is sufficient and in the right format, so

the system can offer the requested functionalities.

First, we need to consider what end-users need, i.e., in which of their tasks do they need support? This highly depends on the application domain. They could, for instance, be in search of a system that automates their recurring daily decisions, so they can focus on more challenging parts of their job. Or they might need a system that supports them in those more challenging tasks. It is important to correctly identify their needs, since this makes a substantial difference in the functionality the system needs to offer.

Second, how can the system support these tasks? Does automating the process flow of such a task (i.e. from input to output) suffice, or are additional functionalities required? For example, the user may be interested in the reasoning process of the system, so explaining this could be an additional functionality. Moreover, users could want a more interactive way to work with the system, e.g. see consequences of a certain choice, specify an output and ask which inputs could have led to that output, or make changes to the underlying model. Different functionalities might require the knowledge model to be formulated differently.

Third, when a system needs to support multiple functionalities, it is unwanted that each of these functionalities requires a different knowledge model. Ideally, we could specify knowledge in such a way that multiple inferences can be executed on the same knowledge base. This principle is called the Knowledge Base Paradigm.

1.2.4 Presentation of Results

Not only should the knowledge in the model be readable, the system should also be able to present the output in such a way that the user can interpret it. When a system has a multitude of different functionalities, of course each of these functionalities should be presented in an interpretable way. This is especially challenging when a user communicates with the system in an interactive way.

1.2.5 Combination of Knowledge Sources

Domain experts typically have a thorough knowledge of their domain. However, they might have some gaps in their knowledge. On the other hand, ML techniques can find patterns in data, but they often miss small nuances that are known by domain experts.

To build a system that has complete knowledge of a certain domain, it is often required to combine knowledge from both sources. There are multiple potential approaches to do this, e.g.:

- Combine the knowledge from both sources into one readable knowledge model.

- Build a model from one knowledge source and use the additional source to fill in the missing knowledge in the model.
- First build two separate knowledge models and combine these two models into one model and use this model as a basis for the functionalities.
- Combine the functionality of two independent systems, each with their own knowledge model. Optionally, these two systems can be adjusted to cooperate.

How this knowledge is combined, highly depends on many factors. It is therefore difficult to find a standardized approach for this. In this thesis, we explore a number of case studies. In each of these cases, we analyse separately which method is best to combine both sources of knowledge.

1.3 Contributions

In this thesis, we focus on the practical applicability of KR techniques. To this end, we examine a number of case studies, which we use to illustrate the structure introduced in Section 1.2. We explore the possibilities to use expert know-how to create a readable model, and how to use data to complement this knowledge if the expert know-how is not sufficient. For this, we selected a number of cases from different domains, with differing requirements and a variety of available knowledge.

In Chapter 2, we propose a framework called cDMN, which expresses constraint knowledge in a readable format. Not only does this notation allow to elicit complex knowledge from experts, it also enables the provision of multiple functionalities on this knowledge.

Chapter 3 focuses on the extraction process of expert know-how and performs a thorough comparison of constraint-based knowledge and deterministic knowledge in terms of readability and functionality. Moreover, we contribute to the development of a general purpose knowledge-based configuration tool. We show that this tool can be used in an industrial context, yielding a practical implementation of the knowledge base paradigm. Additionally, we extend the incomplete knowledge in the knowledge-based system with a number of machine learning techniques, while ensuring a tight integration of both.

In Chapter 4, we explore the probabilistic side of knowledge representation. We present a novel approach to video editing, i.e., a probabilistic forward sampling algorithm to make a video montage in real time. Moreover, we show that even probabilistic knowledge can be expressed in a readable way.

We explore the boundaries of representing knowledge in a readable way in Chapter 5, where we build a readable domain representation starting from data alone, i.e., without any expert know-how.

1.4 Outline of Research

The rest of thesis is structured as follows: each chapter describes a different case study we have conducted.

In Chapter 2 we introduce the cDMN notation, which is able to represent complex constraint-based knowledge in a readable format. We evaluate its applicability by using this notation to model a wide variety of decision modelling problems.

Chapter 3 makes a comparison between deterministic and constraint-based knowledge in the domain of product configuration. It discusses a decision support system that can assist design engineers to create made-to-order machine components. The proposed system offers a multitude of functionalities, which can be used to support the engineers with different aspects of their daily tasks.

The case study discussed in Chapter 4 proposes Probabilistic Logic Programming algorithm automatically produces video montages.

Expert know-how is not always readily available. We investigate this in the context of alarm management in Chapter 5. Here, we use a combination of statistical analysis and ML to respectively identify and investigate interesting alarms.

Finally, in Chapter 6, we conclude the research in this dissertation and indicate which topics could be researched in future work.

Chapter 2

cDMN: A Tight Integration of DMN and Constraint Reasoning

In this chapter, we introduce the cDMN notation, which allows us to represent complex constraint knowledge in a readable format. This notation is based on the DMN-standard, which displays promising results in terms of readability, but is too limiting in two major aspects. First, only simple, deterministic decision-making knowledge can be expressed. More complex problems often require the expression of more advanced knowledge. Second, all currently available solvers are only able to perform forward reasoning, i.e., start from a complete set of inputs and determine the output. We want to be able to use the represented knowledge to offer multiple functionalities. With cDMN, we aim to tackle these shortcomings of DMN.

Figure 2.1 shows which research topics are covered in this chapter. We increase the applicability of DMN by extending the notation to allow more complex knowledge to be expressed. This provides us with a notation that is can represent knowledge in a readable way, which is powerful enough to be applicable in a multitude of domains. This notation enables us to facilitate the knowledge extraction of expert know-how.

Because of the constraint-based nature of the expressed knowledge, cDMN creates the opportunity to reason with this more complex knowledge in different ways. By providing a translation from cDMN to First Order Logic (FO), we can make use of the wide range of inference tasks of the IDP system to provide multiple functionalities.

We use this notation throughout this thesis to improve the readability of models based

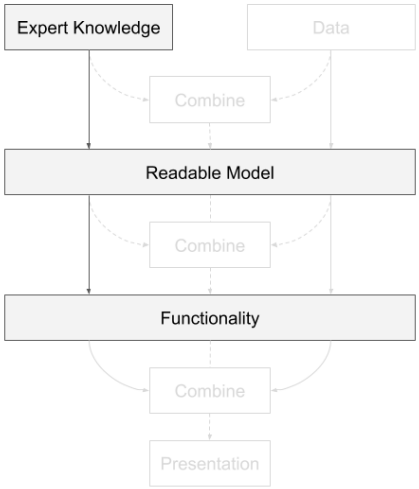


Figure 2.1: Visualization of the research topics in this chapter.

on expert know-how. In Chapter 3, we express product design knowledge in this notation, which greatly improves the readability of the model. In Chapter 4, we explore the possibility to use this notation to represent probabilistic logic.

The contents of this chapter are based on papers published in proceedings of the RuleML+RR 2019 [26] and RuleML+RR 2020 [7] conferences. Preliminary results have been presented on DecisionCamp 2019 [24], and the full notation has been presented on DecisionCamp 2020 [65]. This research has been conducted in collaboration with my colleagues Simon Vandevelde and Marjolein Deryck.

2.1 Introduction

The Decision Model and Notation (DMN) [52] standard, designed by the Object Management Group (OMG), is a way of representing decision logic in a readable, table-based way. It is intended to be used directly by business experts without the help of computer scientists.

While DMN is very effective in modeling deterministic decision processes, it lacks the ability to represent more complex kinds of knowledge. We propose an extension to the DMN standard, called cDMN. It allows more complex problems to be solved, while remaining readable by business users. The main features of cDMN are constraint modelling, quantification, and the use of concepts such as types and functions.

We test the usability of cDMN by modelling a multitude of complex problems. For this, we use the challenges submitted to the Decision Management Community website.[1] The purpose of these challenges is to explore the boundaries of DMN. Community members can then submit a solution, using their preferred decision modelling tools or programming languages. This allows solutions for complex problems to be found and compared across multiple DMN-like representations. So far, none of the available solvers have been able to solve all challenges. Moreover, the available solutions sometimes fail to meet the readability goal of DMN, because the representation is either too complex or too large. We use cDMN to implement solutions to those challenges, to prove that our proposed notation is able to elegantly represent all kinds of knowledge.

This chapter is structured as follows. In Section 2.2.1 we briefly describe the DMN standard. Section 2.3 gives an overview of the challenges used in this chapter. After this, we briefly review the related work in Section 2.4. We discuss both the syntax and semantics of our new notation in Section 2.5. Section 2.6 then briefly discusses the implementation of our cDMN solver. We compare our notation with other notations and evaluate its added value in Section 2.7, and conclude in Section 2.8.

2.2 Preliminaries

2.2.1 DMN

The DMN standard [52] describes the structure of a DMN model. Such a model consists of two components: a Decision Requirements Diagram (DRD), which is a graph that expresses the structure of the model, and Decision Tables, which contain the in-depth business logic. Together, these two components allow for a convenient and user-friendly modelling technology. The DRD allows users to get an overview of the domain at hand, without diving into the specifics of a sub-decision. On the other hand, the individual decision tables are easily understandable with little or no prior training, because they are small and (almost) syntax-less. The DMN standard has been explicitly designed with the goal of being so easy to use that domain experts are able to maintain the logic without the intervention of IT. Hence the expert's ownership of decisions increases, and knowledge can be centralized in one place and continuously updated by domain experts.

Figure 2.2 contains an example of such a decision table. It consists of a number of input columns (darker green) and a single output column (lighter blue). Each row is read as: if the input conditions are met (e.g., if "Age of Person" satisfies the comparison "$\geq$ 18"), then the output expression is assigned the value of the output entry (e.g. "Person

[1] `https://dmcommunity.org/`

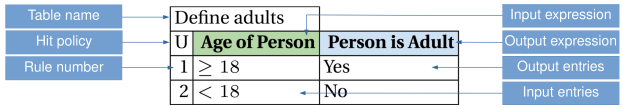


Figure 2.2: Decision table to define whether a person is an adult.

is Adult" is assigned value "Yes"). Only single values, such as strings, numbers and booleans can be used as output entries. In the case where no row matches the input, then each output is either set to the special value *null* (which is typically taken to indicate an error in the specification) or to the output's default value, if one was provided.

The behaviour of a decision table is determined by its hit policy. There are a number of *single hit* policies, which define that a table can have at most one output for each possible input, such as "Unique" (no overlap in input conditions may occur), "Any" (if there is an overlap, the outputs must be the same) and "First" (if there is an overlap, the first applicable row should be selected). *Multiple hit* policies also exist, such as C (collect the output of all applicable rows in a list) and C+ (sum the output of all applicable rows). Regardless of which hit policy is used, each decision table uniquely determines the value of its output(s).

The entries in a decision table are typically written in the (Simple) Friendly Enough Expression Language, or (S-)FEEL, which is also part of the DMN standard. S-FEEL allows to express simple values, lists of values, numerical comparisons, ranges of values and calculations. When the value of a certain column header does not matter for a specific row, a special don't care value "–" can be used. Decision tables with S-FEEL are generally considered quite readable by domain experts.

In addition to S-FEEL statements, DMN also allows more complex FEEL statements in combination with boxed expressions, which is illustrated in Figure 2.3. As can be seen, using FEEL to its full extent greatly increases the complexity of the representation, which makes it unsuitable to be used by domain experts without the aid of knowledge experts.

2.2.2 IDP and FO(·)

The IDP system [21] allows to model knowledge in the FO(·) language. IDP stands for *imperative declarative programming*, which refers to a declarative knowledge base on the one hand, and the ability to apply different inference algorithms to such a knowledge base through an imperative shell on the other hand. As such, it is an implementation of the Knowledge Base Paradigm, which states that the domain knowledge can be separated from the tasks that can be executed on this knowledge to solve problems. The imperative shell makes use of the lua programming language [44]. The declarative



Figure 2.3: Extract of the map colouring solution in standard DMN with FEEL.

knowledge base uses a classical First Order Logic based language, FO(·). The (·) refers to the various extensions of FO that are supported in FO(·): aggregates, types and inductive definitions. The extensions allow us to express complex knowledge, such as the fact that the size of a product equals the sum of the sizes of all components used in the product:

$$\forall p[Product] : \ Totalsize(p) = sum\{c[Component] : \ Used(c) : \ Size(c)\}.$$

Where $x[T]$ denotes a typed variable x of type T, and an expression of the form $sum\{x_1, \ x_2 : \ \varphi(x_1, x_2) : \ F(x_1, x_2)\}$ sums function F applied to all tuples (x_1, x_2) for which the logical formula φ holds.

The knowledge in an IDP specification is structured in three blocks, the *vocabulary*, the *structure* and the *theory*. A *vocabulary* describes the symbols that are used in a logical specification. It is a set of types, functions and predicates (i.e., boolean-valued functions).

A *structure* is an interpretation of a certain vocabulary. As usual, each type is interpreted by a set of domain elements, each n-ary predicate is interpreted by a set of n tuples of domain elements, and each n-ary function by a mapping of n tuples to elements. In addition, FO(·) also allows three-valued interpretations of relations, in which it may be unknown whether a tuple belongs to the relation or not.

A *theory* is a set of logical formulas. When all formulas in the theory (T) hold in a certain structure (S), that structure is a model of the theory ($S \models T$).

Making use of an efficient solver, a number of inference tasks can be performed, on this knowledge, such as *model expansion* (find a more precise two-valued structure

Property	(%)
1. Aggregates needed	57.14
2. Constraints	33.33
3. Universal quantification	28.75
4. Optimization	23.81

Table 2.1: Percentage of occurrence of properties in challenges.

that satisfies the theory), *optimization* (find a more precise two-valued structure that satisfies the theory and optimizes a numerical term), *propagation* (generate all model expansions and combine them into one three-valued structure), and so on. These inference tasks are discussed in more detail in Section 3.5.5.

2.3 Challenges Overview

Of all the challenges on the DM Community website, we selected those for which no straightforward DMN-like solution had been submitted. The list of the 21 challenges that meet this criterion can be found in the cDMN documentation [63].

We categorize these challenges according to four different properties. Table 2.1 shows the list of properties, and the percentage of challenges that have this property.

The most frequent property is the need for aggregates (57.14%), such as counting the number of violated constraints in *Map Coloring with Violations* or summing the number of calories of ingredients in *Make a Good Burger*. The second most frequent property is having constraints in the problem description (33.33%). For example, in *Who Killed Agatha*, the killer hates the victim and is no richer than her; or the constraint in *Map Coloring* states that two bordering countries cannot share the same colour. The next property, universal quantification (28.75%), is that a statement applies to every element of a type. For example, in *Who Killed Agatha?*, nobody hates everyone. The final property, optimization, occurs in 23.81% of the challenges. For example, in *Zoo, Buses and Kids* the cheapest set of buses must be found.

2.4 Related Work

It has been recognized that even though DMN has many advantages, it is somewhat limited in expressivity [15, 26]. This holds especially for decision tables with S-FEEL, the fragment of FEEL that is considered most readable. While full FEEL is more expressive, it is not suitable to be used by domain experts without the aid of knowledge

experts. Moreover, it does not provide a solution to other shortcomings, such as the lack of constraint reasoning and optimization.

One of the systems that does effectively support constraint solving in a readable DMN-like representation is the OpenRules system [53]. It enables users to define constraints over the solution space by allowing "Solver Tables" to be added alongside decision tables. In contrast to standard decisions, which assign a specific value to an output, Solver Tables allow for setting constraints on the output space. OpenRules offers a number of *DecisionTableSolve-Templates*, which can be used to specify these constraints. It is possible to either use these predefined templates, or define such a template manually if the predefined ones are not expressive enough. Even though this system extends the range of applications that can be handled, there are three reasons why it does not offer the ease of use for business users that we are after. First, because of the wide range of available templates for solver tables, which differ from that of standard decision tables, using the OpenRules constraint solver entails a steep learning curve. For example, the OpenRules implementation of *Who Killed Agatha?*[2] not only uses standard decision tables, but also 3 different templates: the "ActionXoperY" template to compare two terms, the "ActionXoperYcompareZ" template that applies a mathematical operator to two terms and compares it to a third term, and the "ActionSum" template that sums a set of numbers and assigns it to a variable. By contrast, cDMN allows the same knowledge to be expressed in one type of constraint table. Second, the solver's functionality can only be accessed through the Java API, which goes against the DMN philosophy [52, p. 15]. Third, because of the lack of quantification in OpenRules, solutions are generally dependent on domain size, which reduces readability and maintainability.

Another system that aims to increase the expressiveness of DMN is Corticon [55]. It implements a basic form of constraint solving by allowing the user to filter the solution space. While this approach indeed improves expressiveness, it decreases readability. Moreover, some constraints can only be expressed by combining a number of rules and a number of filters. For example, when expressing "all female monkeys are older than 10 years", this must be split up in two parts: (1) a rule that states `Monkey.gender = female & Monkey.Age < 10 THEN Monkey.illegal = True` and (2) a filter that states that a monkey cannot be illegal: `Monkey.illegal = False`. There are no clear guidelines about which part of the constraints should be in the filter and what should be a rule.

In [15], Calvanese et al. propose an extension to DMN which enables the expression of additional domain knowledge in Description Logic. They share our goal of extending DMN to express more complex real-life problems. However, they introduce a completely separate Description Logic formalism, which seems too complex for a

[2] `https://openrules.wordpress.com/2014/11/08/decision-determine-a-killer-of-aunt-agatha/`

domain expert to use. Unfortunately, they did not submit any solutions to the DMN Challenges, which leaves us unable to compare its expressiveness in practice.

2.5 cDMN: Syntax & Semantics

While DMN allows users to elegantly represent a deterministic decision process, it lacks the ability to specify constraints on the solution space. The cDMN framework extends DMN, by allowing constraints to be represented in a straightforward and readable manner. It also allows for representations that are independent of domain size by supporting types, functions, relations and quantification.

We now explain both the usage and the syntax of every kind of table present in cDMN.

2.5.1 Glossary

In logical terms, the "variables" used in standard DMN correspond to 0-ary functions (we allow these kinds of symbols to be introduced in the "constant" table). cDMN extends these by adding n-ary functions and n-ary relations. Similarly to OpenRules and Corticon, we allow the user to define their vocabulary by means of a glossary. This glossary contains every symbol used in a cDMN model. It consists of at most five glossary tables, which each enumerate a different kind of symbol. An example glossary for the *Who Killed Agatha?* challenge is given in Figure 2.4.

Type

Name	Type	Values
Person	string	Agatha, Butler, Charles
Number	int	[0..100]

Function

Name	Type
NbOfHatees of Person	Number

Constant

Name	Type
Killer	Person

Relation

Name
Person hates Person
Person is richer than Person

Boolean

Name
Suicide

Figure 2.4: Example glossary in cDMN for the *Who Killed Agatha?* problem.

Type table In the type table, type symbols are declared. The value of each type is a finite set of domain elements, specified either in the glossary or in a data table (see Section 2.5.3). An example is the type `Person`, which contains the names of people. Even though, theoretically, type domains may be (partially) overlapping, we discourage this, as it might introduce ambiguity in the resulting model.

Function table In the function table, a symbol can be declared as a function of one or more types to another. To declare a function f that takes n arguments of types T_1, ..., T_n, the syntax $f of T_1 and \ldots and T_n$ is used. In Figure 2.4, the `NbOfHatees of Person` is declared as a function that maps a person to a number. In constraint or decision tables, the same syntax can then later be used to actually apply the function to a tuple of arguments, e.g., *NbOfHatees of Agatha* represents the number that results from applying the function *NbOfHatees* to the argument *Agatha* of type *Person*.

Constant table For each domain element, a constant with the same name is automatically introduced, which allows the user to refer to this domain element in constraint or decision tables. For instance, the user can use the constant `Agatha` to refer to the domain element `Agatha`. In addition, the constant table allows other constants to be introduced. Recall that such logical constants correspond to standard DMN variables. In our example case, we use a constant `Killer` of the type `Person`, which means it can refer to any of the domain elements `Agatha`, `Butler` or `Charles`.

Relation table In the relation table, a verb phrase can be declared as a relation on one or more given types. For instance, the relation `Person is Adult` denotes for each `Person` whether they are an adult. This relation translates to the unary predicate $isAdult$. n-ary relations can be defined by using *n* arguments in the name, e.g. `Person is richer than Person` is a relation with two arguments (both of the type `Person`), that denotes whether one person is richer than another.

Boolean table The boolean table contains boolean symbols (i.e. propositions), which are either true or false. An example is the boolean `Suicide`, which denotes whether the murder is a suicide.

2.5.2 Decision Tables and Constraint Tables

As stated earlier in Section 2.2.1, a standard decision table uniquely defines the value of its outputs. When no single row is applicable, the output of the table is forced to *null* by default. We extend DMN by allowing a new kind of table, called a *constraint table*, which does not have this property.

Whereas decision tables only allow single values to appear in output columns, our constraint tables allow arbitrary S-FEEL expressions in output columns. Each row of a constraint table represents a logical *implication*. A constraint table is satisfied when, for all rows, if the input conditions are met, the output conditions are also met. This

means that in constraint tables, if, for instance, none of the rows are applicable, the outputs can take on an arbitrary value, as opposed to being forced to *null*. In constraint tables, no default values can be assigned. Because of these changes, a set of cDMN tables does not define a single solution, but rather a solution space containing a set of possible solutions.

We introduce a new *hit policy* to identify constraint tables. We call this the *Every* hit policy, denoted as *E**, because it expresses that every implication in the table must be satisfied. An example of this can be found in Figure 2.5, which states that each person hates less than 3 people.

cDMN does not only introduce constraint tables, it also extends the expressions that are allowed in column headers, both in decision and constraint tables. Such a header can consist of the following expressions:

- A type $Type$. Such expression introduces a new variable $Type$ of type $Type$, which is only defined in the scope of the table.
- An expression of the form "$Type$ called $name$". This expression introduces a new variable $name$ of type $Type$ in the scope of the table.
- A constant.
- An expression of the form "$Function$ of arg_1 and ... and arg_n", where each of the arg_i is another header expression or a previously introduced variable. This expression applies $Function$ to its arguments.
- An arithmetic combination of header expressions (such as a sum of constants).

The first two kinds of expressions are called *variable* header expressions. They allow *universal quantification* in cDMN. Each input column whose header consists of such a *variable* expression either introduces a new universally quantified variable (we call this a *variable-introducing* column), or refers back to a variable introduced in a preceding *variable-introducing* column. Subsequent, left-to-right, uses of the same type name (in case of the first kind of variable-introducing expression) or of the variable name (in case of the second kind) then refer back to this universally quantified variable. Whenever a type or variable name appears in a header of a column that is itself not variable-introducing, a unique preceding variable-introducing column that has introduced this variable must exist.

In short, constraint and decision tables that include (multiple) variable introducing columns, should be satisfied for each combination of the values of the introduced variables' types.

The table in Figure 2.5 shows an example of quantification in cDMN. This table introduces a universally quantified variable of the type *Person*, places no restrictions

on this variable (i.e., "–"), and hence states that each person should hate less than three others.

To illustrate the use of named variables, Figure 2.6 defines variables `c1` and `c2`, both of the type `Country`, and states that when those countries share a border, they cannot have the same colour.

Currently, the cDMN notation does not provide a specific construct for existential quantification.

In summary, this section has discussed three ways in which cDMN extends DMN. First, the hit policy *E** changes the semantics of the table. Second, constraint tables allow S-FEEL expressions in the output columns. Third, cDMN allows universal quantification, functions, predicates and calculations to be used in both decision tables and constraint tables.

Noone hates all		
E*	**Person**	**NbOfHatees of Person**
1	–	< 3

Figure 2.5: Implementation of "Nobody hates everyone" in *Who Killed Agatha?*.

Bordering countries cannot share colours				
E*	**Country called c1**	**Country called c2**	**c1 and c2 are Bordering**	**Colour of c1**
1	–	–	Yes	Not(Colour of c2)

Figure 2.6: Constraint table with quantification in cDMN, defining that bordering countries cannot share colours.

2.5.3 Data Tables

Typically, problems can be split up into two parts: (1) the general logic of the problem, and (2) the specific problem instance that needs to be solved. Take for example the map colouring problem: the general logic consists of the rule that two bordering countries cannot share a colour, whereas the instance of the problem is the specific map (e.g. Western Europe) to colour. cDMN extends the DMN standard to include *data tables*, which are used to represent the problem instances, separating them from the general logic. The format of a data table closely resembles that of a decision table, with a couple of exceptions. Instead of a hit policy, a data table has "data table" in its name. Furthermore, only basic values, i.e., domain elements (strings), integers and floats are allowed in data tables. It is also possible for columns to have more than one value in a certain row, in which case the row is instantiated for the combination of each of the values of the columns. As an example, a snippet of the data table for the *Map Coloring* challenge is shown in Figure 2.7. This use of data tables offers two advantages.

1. There is a methodological advantage: by separating the data tables from the decision tables, reusing the specification becomes easier.

2. If the user chooses to enumerate the domain of a type in the glossary, then the system checks that each value in a data table indeed belongs to the domain of the appropriate type. This helps to prevent typographical and other errors in the input data or glossary. If the user chooses not to enumerate a type in the glossary, then the type's domain defaults to the set of all values in the data table.

Data Table: Declaring which countries border			
	Country called c1	**Country called c2**	**c1 and c2 are Bordering**
1	Belgium	France, Luxembourg, Netherlands, Germany	Yes
2	Germany	France, Denmark, Luxembourg, Belgium, Netherlands	Yes

Figure 2.7: Data table describing countries and their neighbours.

2.5.4 Execute Table

A standard DMN model defines a deterministic decision procedure. It is always used in the same way: the external inputs are supplied by the user, after which the values of the output variables are computed by forward propagation.

In cDMN, this is no longer the case. We can fill in as many or as few variables as we want, and use the model to derive useful information about the not-yet-known variables. By employing an *execute table*, users can specify what the model is to be used for: model expansion or optimization. Model expansion creates a given number of solutions, the syntax is "get n models", where n is the amount of requested models. Optimization looks for the solution with either the lowest or highest value for a given numerical term. Examples of such execute tables can be found in Figure 2.8 and 2.13.

2.5.5 Semantics of cDMN

We describe the semantics of cDMN by translating it to the FO(·) language used by the IDP system [11, 69]. FO(·) is a rich extension of First Order Logic, adding concepts such as types, aggregates and inductive definitions. The semantics of cDMN is defined by the semantics of each of its sub-components.

Execute
Minimize Cost

Execute
Get 5 models

Figure 2.8: Examples of execute tables.

It is straightforward to translate the glossary into an FO(·) vocabulary: types, functions, constants, relations and booleans are each translated to their FO(·) counterpart.

Decision tables retain their usual semantics as described by Calvanese [15]. We briefly recall this semantics. Each cell of a decision table (i, j) corresponds to a formula $F_{ij}(x)$ in one free variable x. For instance, a cell "≤ 50" corresponds to the formula "$x \leq 50$". A decision table with the *unique* or *any* hit policy with rows R, input columns I and output columns O is a conjunction of material implications:

$$\bigwedge_{i \in R} \left(\bigwedge_{j \in I} F_{ij}(H_j) \Rightarrow \bigwedge_{k \in O} F_{ik}(H_k) \right) \tag{2.1}$$

where H_j is the header of column j. When a decision table is incomplete, it is possible no rows match. In this case, the output given a special *null* value, i.e.,

$$\left(\bigwedge_{i \in R} \neg \bigwedge_{j \in I} F_{ij}(H_j) \right) \Rightarrow \bigwedge_{k \in O} H_k = null \tag{2.2}$$

In short, a decision table is satisfied when, for each row of which all the input conditions are met, all output conditions are also met. When no rows are applicable, the output is forced to *null*.

For example, in standard DMN (i.e., without the addition of function expressions), the table in Figure 2.2 corresponds to the logical formula $(AgeOfPerson \geq 18 \Rightarrow PersonIsAdult = Yes) \wedge (AgeOfPerson < 18 \Rightarrow PersonIsAdult = No)$.

Data tables are simply specific cases of decision tables.

The semantics of simple constraint tables (without quantification and functions) is also a conjunction of implications, as we described in [26]. The semantics of constraint tables and decision tables differ in the interpretation of incomplete tables: when no rows are applicable in decision tables, the output is forced to *null* (i.e., the implicit default value is *null*), while the output in constraint tables can take any arbitrary value.

We now extend this semantics to take variables and quantification into account. Our first step is to define a function that maps cDMN expressions to terms. For the most part, this definition corresponds to that of Calvanese [15].

Similarly to Calvanese, we translate each entry c in a cell (i, j) of a table into a formula $F_{ij}(x)$ in one free variable x:

- If c is of the form "θe" with θ one of the relational operators $\{\leq, \geq, =, \neq\}$, then $F_{ij}(x)$ is the formula $x\ \theta\ t(e)$;

- If c is of the form $Not\ e$, then $F_{ij}(x)$ is $x \neq t(e)$;
- If c is a list $e_1, \ldots, e_n$, then $F_{ij}(x)$ is $x = t(e_1) \vee \ldots \vee x = t(e_n)$. As a special case, if c consists of a single expression e, then $F_{ij}(x)$ is $x = t(e)$.
- If c is a range, e.g. $[e_1, e_2)$, then $F_{ij}(x)$ is $x \geq t(e_1) \wedge x < t(e_2)$.

We extend it to take into account the fact that certain expressions – which we call *variable expressions* – must be translated to FO variables. There are three kinds of variable expressions, as we described in Section 2.5.2. We now define a mapping ν that maps each of these three kinds of cDMN variable expressions to a typed FO($\cdot$) variable x of type T, which we denote as $x[T]$:

- The name T of a type is a variable expression. We define $\nu(T) = x_T[T]$, with x_T a new variable of type T.
- An expression e of the form "$Type$ called v" is a variable expression. We define $\nu(e) = v[Type]$.
- If the header of a column contains an expression "$Type$ called v", then v is a variable expression in all subsequent columns of the table and in its body. We define $\nu(v)$ as $v[Type]$.

Such a variable expression introduces a new variable in the scope of the table at hand. Given this function ν, we now define the following mapping $t_\nu(\cdot)$ of cDMN expressions to terms.

- The interpretation of a constants, integer or floating point number expression is the constant or number itself. I.e., for a constant c, $t_\nu(c) = c$; similarly, an for integer or floating point number n, $t_\nu(n) = n$;
- For an arithmetic or other expression e of the form $e_1 \theta e_2$ with $\theta \in \{+, -, *, /, <, >, \leq, \geq, =, \neq, \vee, \wedge\}$, we define $t_\nu(e) = t_\nu(e_1)\ \theta\ t_\nu(e_2)$; In other words, the interpretation of such an expression is the expression applied to the interpretation of its sub-parts.
- The interpretation of a variable expression is the variable itself, i.e., for a variable expression v, we define $t_\nu(v) = \nu(v)$.
- The interpretation of a function expression is that function applied to the interpretation of each of its arguments. For a function expression "$Function$ of arg_1 and ... and arg_n", we define $t_\nu(X) = Function(t_\nu(arg_1), ..., t_\nu(arg_n))$.

We are now ready to define the semantics of a constraint table. If I is the set of input columns of the table, O the set of output columns and $V \subseteq I$ the set of variable introducing columns, we define the semantics of the table T as the following formula ϕ_T:

$$\underset{l \in V}{\forall} \nu(H_l) : \bigwedge_{i \in R} \Big(\bigwedge_{j \in I} t_\nu\big(F_{ij}(H_j)\big) \Rightarrow \bigwedge_{k \in O} t_\nu\big(F_{ik}(H_k)\big) \Big) \tag{2.3}$$

In other words, for each expansion of the introduced variables' types, all table rows should be satisfied. Such a row is satisfied when, if all input conditions are met, all its output conditions are also met.

For example, in Figure 2.5, $\nu(H_1) = x[Person]$ and $t_\nu(H_1) = x$, $t_\nu(H_2) = NbOfHatees(t_\nu(H_1)) = NbOfHatees(x)$, which leads to the formula:

$$\forall x[Person] : NbOfHatees(x) < 3.$$

which states that each person x hates less than 3 people.

We prove that this semantics is also valid for regular DMN tables. In this case, no variables are introduced, thus $V = \emptyset$, and only *constant* symbols are allowed, so $t_\nu(F_{ij}(H_j)) = F_{ij}(H_j)$. As a result, the formula in Equation 2.3 simplifies to that in Equation 2.1.

Decision tables with multiple hit policies have different semantics. We first describe the semantics of *C+*, *C<* and *C>* tables, which are almost identical. We define the semantics of a *C+* table with one output header H_k:

$$\underset{w \in W}{\forall} \nu(H_w) : t_\nu(H_k) = \sum_{i \in R} sum\big\{\bar{x} : \bigwedge_{j \in I} t_\nu\big(F_{ij}(H_j)\big) : t_\nu(F_{ik}(\bar{x}))\big\} \tag{2.4}$$

Where $W \subseteq V$ is the subset of variable introducing columns V of which the variable appears in the output header $t_\nu(H_k)$, $\bar{x}$ is a tuple of new variables of the set of the remaining variable introducing columns $U = V \setminus W$, and $sum\{\bar{x} : \varphi(\bar{x}) : F(\bar{x})\}$ denotes the sum of all function values F applied to the tuples $\bar{x}$ for which φ holds.

This can be interpreted as follows: First, when no variables are introduced (i.e., $U = V = W = \emptyset$), this formula sums the output values for each of the rows that meet the input criteria. This is precisely the definition of a standard DMN *C+* table.

Second, when variables are introduced in a table, but the output header contains no variables ($W = \emptyset$), it is assigned the sum of each row's value. This value is the sum of the output function values for each of the variable expansions, given that they satisfy the corresponding input conditions.

Charge of Person				
C+	**Person**	**Item**	**Item in basket of Person**	**Charge of Person**
1	–	–	Yes	Price of Item

Figure 2.9: Decision table that determines the charge of a person, based on the contents of their shopping basket.

Third, when the output header does contain variables, each expansion of the header expression is assigned the sum of the output function values for all applicable rows and expansions of the remaining variables.

We illustrate this semantics with an example. In the decision table of Figure 2.9: $W = \{Person\}$, $\nu(H_w) = p[Person]$, $t_\nu(H_k) = Charge(p)$, $\bar{x} = y[Item]$, $t_\nu(F_{1k}(\bar{x})) = Price(y)$ and $\bigwedge_{j \in I} t_\nu(F_{ij}(H_j)) = InBasket(y, p)$. This results in the logical sentence:

$$\forall\, p[Person] : Charge(p) = sum\{(Item) : InBasket(Item, p) : Price(Item)\}$$

The table in Figure 2.9 determines the $charge$ of a $Person$, therefore, the logical sentence starts with quantifying a variable of the type $Person$. For each $Person$, their $Charge$ can be calculated by summing the *price* of all *item*s that are in their shopping basket.

The semantics of *C<* and *C>* are defined analogously, where, instead of summing all values, the minimum and maximum value is selected respectively.

Decision tables with a *C#* hit policy have a slightly different semantics, i.e..

$$\bigvee_{w \in W} \nu(H_w) : t_\nu(H_k) = \left|\left\{x \mid \exists_{u \in U}\, \nu(H_u) : \bigvee_{i \in R} \left(t_\nu(F_{ik}(x)) \wedge \bigwedge_{j \in I} t_\nu(F_{ij}(H_j))\right)\right\}\right| \tag{2.5}$$

Where U and W are defined analogously as in Equation 2.4.

This can be interpreted as follows: First, in the most simple case, when no quantification is used, this formula counts the number of rows (with distinct outputs) that meet the input criteria. This is the same behaviour as a standard DMN *C#* table.

Second, when the output header contains no variables, it is assigned the number of distinct values in the output column (given that the input conditions are met), after the variables have been expanded. Indeed, in this case, $W = \emptyset$ and $U = V$. The output header is assigned the size of the set x given that there exists an expansion of variables for which one of the rules that has x as output fires.

Third, when the output header does contain variables, each expansion of the header expression is assigned the number of distinct values that occur in the output column (given that the input conditions are met) for that expansion of the header expression.

Charge of Person				
$C\#$	**Person**	**Person is Friend**	**Person is Family**	**NbInvitations**
1	–	Yes	–	Person
2	–	–	Yes	Person
3	–	–	Yes	Spouse of Person

Figure 2.10: Decision table that counts the number of invited guests.

For instance, in the decision table in Figure 2.10: $W = \emptyset$, $t_\nu(H_k) = NbInvitations$, $U = \{Person\}$, $\nu(H_u) = p[Person]$. For the first row, $\big(t_\nu(F_{1k}(x)) \wedge \bigwedge_{j\in I} t_\nu(F_{1j}(H_j))$ is equivalent to $x = p \wedge isFriend(p)$, the second row is defined analogously, and the in the third row, $t_\nu(F_{3k}(x)) \wedge \bigwedge_{j\in I} t_\nu(F_{3j}(H_j))$ translates to $x = Spouse(p) \wedge isFamily(p)$. Consequently, the table in Figure 2.10 is logically equivalent to:

$$\begin{aligned} NbInvitations = \big|\{x \mid \exists p[Person] :& \\ & x = p \wedge isFriend(p) \vee \\ & x = p \wedge isFamily(p) \vee \\ & x = Spouse(p) \wedge isFamily(p) \\ \}\big| & \end{aligned} \tag{2.6}$$

In the table of Figure 2.10, the output header is a constant ($NbInvitations$), therefore no quantification is required. The value of this constant is calculated as the number of persons that are either friends, family or the spouse of family, while ensuring that duplicate persons (e.g., friends that are also family) are not counted multiple times.

The above transformation turns each decision or constraint table T into an FO($\cdot$) formula ϕ_T. The glossary and data tables together define a structure S for part of the vocabulary. The domain of S consists of the union of the interpretations I_t of all the types t. If t is enumerated in the glossary, then I_t is this enumeration. Otherwise, I_t consists of all the values that appear in a data table in a column of type t. The structure S interprets all the relations, functions, constants and booleans for which a data table is provided, and it interprets them by the set of tuples that is given in this table.

The set of "solutions" of a cDMN model is the set $ModelExpand(\Phi, S)$ of all models that expand the structure S w.r.t. the theory $\Phi = \{\phi_T \mid T \text{ is a constraint or decision table}\}$, i.e., the set of all structures $S' \models \Phi$ that extend S to the entire vocabulary.

2.6 Implementation

This section gives a very brief overview of the inner workings of the cDMN solver [64]. The solver consists of two parts: a constraint solver (the IDP system [14]), and a converter from cDMN to IDP input. In principle, any constraint solver could be used, but we chose the IDP system because of its flexibility.

The cDMN to IDP converter was built using Python3, and works in a two-step process. It first interprets all tables in a `.xlsx` sheet and converts them into Python objects. All glossary tables are converted into a single `Glossary` object, containing `Type`, `Function`, `Predicate`, etc. objects. The constraint and decision tables are then evaluated individually. A lex/yacc parser inspects each cell and parses it to an arithmetic (or logical) formula of cDMN expressions, such as "$Function$ of arg_1 and arg_2". Such cDMN expressions are then interpreted using the constructed `Glossary` object into a FO($\cdot$) expression, e.g., "$Function(arg_1, arg_2)$". Each cDMN table is then converted to a FO($\cdot$) formula, as described in Section 2.5.5.

The created Python objects are then converted into IDP blocks. More detailed information about this conversion can be found in the cDMN documentation [63], along with an explanation of the usage of the solver and concrete examples of cDMN implementations.

2.7 Results and Discussion

In this section we first look at three of the DM Community challenges, each showcasing a feature of cDMN. For each challenge, we compare the DMN implementations from the DM Community website with our own implementation in cDMN. Afterwards, we compare all challenges on size and quality.

2.7.1 Constraint Tables

Constraint tables allow cDMN to model constraint satisfaction problems in a straightforward way. For example, in *Map Coloring*, a map of six European countries must be coloured in such a way that no neighbouring countries share the same colour. For this challenge, a pure DMN implementation was submitted, of which Figure 2.3 shows an extract. The implementation uses complicated FEEL statements to solve the challenge. While these statements are DMN-compliant, they are nearly impossible for a business user to write without help. In cDMN, we can use a single straightforward constraint table to solve this problem, as shown in Figure 2.6. Together with

DecisionTable SituationRules					
ConditionVarOperValue			ConclusionVarOperValue		
String attribute	Oper op	String value	String attribute	Oper op	String value
...	...	...	...	...	...
Mike's Resting Place	=	Rock	Mike's Fruit	=	Apple
Sam's Resting Place	=	Rock	Sam's Fruit	=	Apple
Anna's Resting Place	=	Rock	Anna's Fruit	=	Apple
Harriet's Resting Place	=	Rock	Harriet's Fruit	=	Apple
...	...	...	...	...	...

(a) Open Rules

Monkey Constraints			
E*	**Monkey**	**Place of Monkey**	**Fruit of Monkey**
...	...	...	...
2	–	Rock	Apple
...	...	...	...

(b) cDMN

Figure 2.11: Extract of *Monkey Business* implementation in (a) OpenRules and (b) cDMN, specifying "The monkey who sits on the rock is eating the apple".

the glossary and a data table (Figure 2.7), this forms a complete yet simple cDMN implementation.

2.7.2 Quantification

Quantification is useful in the *Monkey Business* challenge. In this challenge, we want to know for four monkeys what their favourite fruit and their favourite resting place is, based on some information. There are two DMN-like submissions for this challenge: one using Corticon, and one using OpenRules.

One of the pieces of information is: `The monkey who sat on the rock ate the apple.` The OpenRules implementation has a table with a row for each monkey, which states that if this monkey's resting place was a rock, their fruit was an apple (Figure 2.11a). In other words, for n monkeys, the OpenRules implementation of this rule requires n lines. Because of quantification, cDMN requires only one row, regardless of how many monkeys there are (Figure 2.11b). The Corticon implementation also uses a similar quantification for this rule.

Another rule states that no two monkeys can have the same resting place or fruit. In both the Corticon and OpenRules implementations, this is handled by two tables with a row for each pair of monkeys. The Corticon tables are shown in Figure 2.12a. Each row either states that two monkeys have different fruit, or that they have different place. Therefore, n monkeys require $\frac{n\times(n-1)}{2}$ rows. By contrast, the cDMN implementation seen in Figure 2.12b requires only a single row to express the same.

Mike.fruit<>Sam.fruit	T
Mike.fruit<>Harriet.fruit	T
Mike.fruit<>Anna.fruit	T
Sam.fruit<>Harriet.fruit	T
Sam.fruit<>Anna.fruit	T
Harriet.fruit<>Anna.fruit	T

Mike.place<>Sam.place	T
Mike.place<>Harriet.place	T
Mike.place<>Anna.place	T
Sam.place<>Harriet.place	T
Sam.place<>Anna.place	T
Harriet.place<>Anna.place	T

(a) Corticon

Different Preferences				
E*	**Monkey called m1**	**Monkey called m2**	**Place of m1**	**Fruit of m1**
1	–	not(m1)	not(Place of m2)	not(Fruit of m2)

(b) cDMN

Figure 2.12: Extract of the *Monkey Business* implementation in (a) Corticon and (b) cDMN, defining that no monkeys share fruit and no monkeys share the same place.

2.7.3 Optimization

In the *Balanced Assignment* challenge, 210 employees need to be divided into 12 groups, so that every group is as diverse as possible. The department, location, gender and title of each employee is known. This is quite a complex problem to handle in DMN. As such, of the four submitted solutions, only one was DMN-like: an OpenRules implementation, using external CP/LP solvers. The logic for these external solvers is written in Java. Although the code is fairly compact, it cannot be written without prior programming knowledge. Because optimization is built-in in cDMN, we can solve the problem with two decision tables and one constraint table. The table *Diversity score*, shown in Figure 2.13, adds 1 to the total diversity score if two similar people are in a different group. Maximizing this score then results in the most diverse groups.

Diversity score								
C+	**Person called p1**	**Person called p2**	**Department of p1**	**Location of p1**	**Gender of p1**	**Title of p1**	**Group of p1**	**Score**
1	–	–	= Department of p2	–	–	–	not(Group of p2)	1
2	–	–	–	= Location of p2	–	–	not(Group of p2)	1
3	–	–	–	–	= Gender of p2	–	not(Group of p2)	1
4	–	–	–	–	–	= Title of p2	not(Group of p2)	1

Execute
Maximize Score

Figure 2.13: Decision tables and constraint table for Balanced Assignment.

	Who Killed Agatha?	Change Making Decision	Make a Good Burger	Duplicate Product Lines	Collection of Cars	Monkey Business	Vacation Days	Family Riddle	Greeting a Customer	Online Dating Services	Classify Employees	Reindeer Ordering	Zoo, Buses and Kids	Balanced Assignment	Vacation Days Advanced	Map Coloring	Map Coloring Violations	Crack the Code	Numerical Haiku
cDMN	53	26	35	20	26	47	38	76	88	45	36	14	24	55	124	21	48	77	41
Corticon	54	14	20	19	45	64	32	22		78	21	64							
OpenRules	176		95	21		150	31		205		70	111	43	30	97				
Others			76[1]		48[1]		14[2]				34[3]	370[4]				31[4]			

Table 2.2: Comparison of the number of cells used per implementation. Other implementations: 1. FEEL, 2. Blueriq, 3. Trisotech, 4. DMN

2.7.4 Overview of Challenges

Of the 21 challenges we considered, cDMN is capable of successfully modelling 19. In comparison, there were 12 OpenRules implementations and 12 Corticon implementations submitted. Note that we have not examined whether OpenRules and Corticon might be capable of modelling more challenges than those for which a solution was submitted.

To compare cDMN to other approaches, we focus on two aspects: quantity (how big are they?) and quality (how readable and how scalable are they?). The size of implementations was measured by counting the number of cells used in all the decision logic tables. Each solver has a distinct notation for meta information (e.g., a glossary in cDMN), some of which are not table-based. Moreover, in most of our implementations we used data tables to provide input data, whereas the other solvers have different mechanisms for this. Consequently, it is difficult to compare the size of these aspects of the solutions. Therefore, we exclude glossaries, data tables and their counterparts in other solvers from this measure. Table 2.2 shows that cDMN and Corticon alternate between having the fewest cells, and that OpenRules usually has the most. In general, OpenRules implementations require many cells because each cell is very simple. For instance, even an "=" operator is its own cell. The Corticon implementations, on the other hand, contain more complex cells, rendering them more compact.

Because of this, OpenRules implementations are usually easier to read than their Corticon counterparts. An example comparison between cDMN and Corticon can be seen in Figure 2.14. Each figure shows a snippet of their *Make a Good Burger* implementation, in which the food properties of a burger are calculated. While the Corticon implementation is more compact, it is less interpretable, less maintainable and dependent on domain size. If the user wants to add an ingredient to the burger,

complex cells need to be changed. In cDMN, we introduce a type *Ingredient*, a number of functions such as *Amount of Ingredient* and *Fat of Ingredient*, and calculate the constant *Total Fat* as the product of the fat in a specific ingredient and the used amount of that ingredient. This enables the user to simply add new ingredients or change the nutritions in the data table and no additional changes need to be done to the model. The OpenRules implementation is fairly readable and modular too, however, it requires a custom scalar product decision table.

	Conditions	1
a	beefPatties.count * 50 + bun.count * 330 + cheese.count * 310 + onion.count + ketchup_lettuce.count * (3+160) + pickle_tomato.count * (260+3)	< 3000
b	beefPatties.count * 17 + bun.count * 9 + cheese.count * 6 + onion.count*2	< 150
c	beefPatties.count * 220 + bun.count * 260 + cheese.count * 70 + onion.count*10 + ketchup_lettuce.count * (4+20) + pickle_tomato.count * (5+9)	< 3000

(a) Corticon

DecisionTable DefineBurgerTotals

ActionScalProd

Scalar Product Name	Coefficients	Variables
Total Sodium	50,330,310,1,260,3,160,3	Beef Patty Items,Bun Items,Cheese Items,Onions Items,Pickles Items,Lettuce Items,Ketchup Items,Tomato Items
Total Fat	17,9,6,2,0,0,0,0	Beef Patty Items,Bun Items,Cheese Items,Onions Items,Pickles Items,Lettuce Items,Ketchup Items,Tomato Items
Total Calories	220,260,70,10,5,4,20,9	Beef Patty Items,Bun Items,Cheese Items,Onions Items,Pickles Items,Lettuce Items,Ketchup Items,Tomato Items
Total Cost	25,15,10,9,3,4,2,4	Beef Patty Items,Bun Items,Cheese Items,Onions Items,Pickles Items,Lettuce Items,Ketchup Items,Tomato Items

(b) OpenRules

Determine Nutrition

C+	Ingredient called I	Total Sodium	Total Fat	Total Calories	Total Cost
1	–	Amount of I * Sodium of I	Amount of I * Fat of I	Amount of I * Calories of I	Amount of I * Cost of I

Nutrition Constraints

E*	Total Sodium	Total Fat	Total Calories
1	<3000	<150	<3000

(c) cDMN

Figure 2.14: Calculating the food properties of burgers in (a) Corticon, (b) OpenRules and (c) cDMN.

A comparison between cDMN and OpenRules can be found in Figure 2.15a and 2.15b. Here we show a part of their implementations of the *Who Killed Agatha?* challenge. They both show a translation of the following rule: "A killer always hates, and is no richer than, his victim." By using constraints and a constant (`Killer`), cDMN allows us to form a more readable and more scalable table. If the police ever find a fourth suspect, they can easily add the person to the data table without needing to change anything else.

In Section 2.3, we identified four different problem properties. We now demonstrate

DecisionTable KillerHatesAndNoRicherHisVictim						
ConditionXoperY			ActionXoperY			
IF			THEN			
BUTLER KILLED AGATHA	=	1	Butler Hates Agatha	=	1
BUTLER KILLED AGATHA	=	1	Butler Richer Than Agatha	=	0
CHARLES KILLED AGATHA	=	1	Charles Hates Agatha	=	1
CHARLES KILLED AGATHA	=	1	Charles Richer Than Agatha	=	0
AGATHA KILLED AGATHA	=	1	Agatha Hates Agatha	=	1
AGATHA KILLED AGATHA	=	1	Agatha Richer Than Agatha	=	0

(a) OpenRules

Killer constraints		
E*	**Killer hates Agatha**	**Killer richer than Agatha**
1	Yes	No

(b) cDMN

Figure 2.15: Implementation of "A killer always hates and is no richer than their victim" in (a) OpenRules and (b) cDMN

that each property is tackled more easily by one or more of the additions cDMN proposes.

Aggregates needed Figure 2.14c shows how aggregates are both more readable and scalable when using quantification. Moreover, cDMN allows the use of aggregates for more complex operations such as optimization or defining constraints. Furthermore, by allowing a more expressive data representation (such as types and functions), aggregates can be used in a more flexible manner.

Constraints Constraints can be conveniently modelled by constraint tables, such as the constraints in Figure 2.15b, which state that the killer hates Agatha, but is no richer than her. The addition of constraint tables allows for an obvious translation from the statement in natural language to the table.

Universal quantification Problems which contain universal quantification can be compactly represented, as can be seen in a number of figures, such as Figure 2.5, 2.6 and 2.14c.

Optimization Because cDMN directly supports optimization, problems containing this property are easily modelled. Furthermore, by allowing a more expressive data representation (such as types and functions), optimization terms can be defined in a

Property	cDMN answer
Aggregates needed	Quantification, expressive data
Constraints	Constraint tables, quantification, expressive data
Universal quantification	Quantification
Optimization	Optimization

Table 2.3: Comparison between the problem properties and their cDMN answers.

more flexible manner. An example can be found in *balanced assignment* in Figure 2.13. A summary of each problem property and its cDMN answer can be found in Table 2.3.

2.8 Conclusions and Future Work

This chapter introduces a notation to express complex knowledge in a readable format. It is based on the DMN-standard, but it is able to solve more complex problems while still maintaining DMN's level of readability. This extension adds constraint modelling, more expressive data representation (such as types and functions) and quantification.

Constraint modelling allows a user to define a solution space instead of a single solution. A user can generate a desired number of models, or generate a model that optimizes the value of a specific term. Unlike DMN, which only knows constants, cDMN also supports the use of functions and predicates, which allow for more flexible representations. Together with quantification, this allows tables to be constructed in a compact and straightforward manner, while being independent of the size of the problem. This improves readability, maintainability and scalability of tables.

By comparing our cDMN implementations to the implementations of other state-of-the-art DMN-like solvers, we can conclude that cDMN succeeds in increasing expressivity while retaining the simplicity of standard DMN.

In future work it would be interesting to further extend the cDMN notation to be able to represent disjunctions in the output of a constraint table, existential quantification, quantification in output columns and increase compactness of the created models (e.g. currently for each variable, a new column should be introduced). Furthermore, it would be interesting to test this notation in a number of additional use-cases to test its applicability in a multitude of domains.

Chapter 3

Machine Component Design

This chapter discusses the creation of a decision support system assists design engineers in the creation of made to order machine components. The research in this chapter covers each of the topics outlined in Figure 3.1.

Knowledge Sources In this project, both expert know-how and data are used to create a workable system. Over the years, the company's engineers have developed a thorough knowledge of component design. In order to secure the expertise of the most experienced design engineers, their knowledge is extracted during a number of knowledge elicitation workshops, and formulated in a knowledge base. On the other hand, throughout the years, the company kept a log of all designed products in the past and we make use of this historical database to complement the constructed knowledge base.

Readable Model As a first step, we use the readable DMN notation as a knowledge elicitation tool to extract the design knowledge of engineers in a number of workshops. Later in this chapter, we show that the cDMN notation introduced in Chapter 2 can be used to represent a more expressive variant of the knowledge in the DMN model, to solve of a number of issues.

Functionality Because of DMN's limited expressivity, we found that knowledge expressed in this notation cannot be used to provide sufficient functionalities to support the engineers in their daily design tasks. One example of this is the absence of support for alternative solutions. Therefore, we express this design knowledge in a constraint-based way, first in First Order Logic (FO), and next in cDMN. We then use

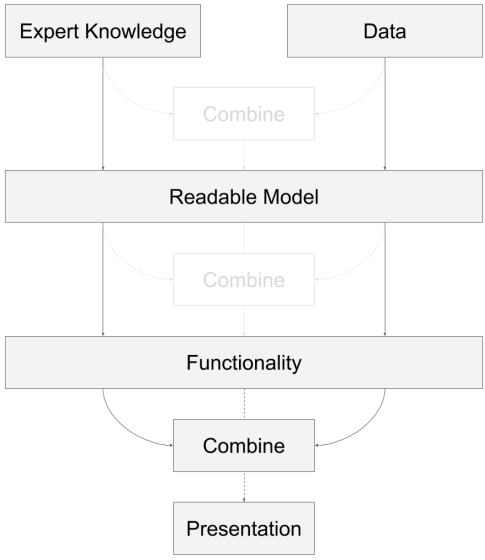


Figure 3.1: Visualization of the research topics in this chapter.

the IDP system to reason with this model. This system implements the knowledge base paradigm, which means it offers a multitude of inferences that can be performed on the same knowledge base. These inferences can be used to provide the necessary functionalities to support the engineers in multiple aspects of their design tasks.

In addition to these functionalities based on expert knowledge, the image recognition and clustering algorithms described by Van Daele and Meert [62] can be used to provide additional functionalities based on historically collected data.

Combining Knowledge Sources In this chapter, we combine the expert know-how and data-based knowledge by combining and aligning the separate functionalities of each of the models. This ensures that both sources of knowledge can be used to their full potential in terms of functionality. Moreover, the tight integration of the two components assures a pleasant user-experience.

Presentation We propose an extension to the IDP AutoConfig tool presented in [20]. This tool allows the engineers to communicate interactively with the represented knowledge. We add a number of requested functionalities and ensure the tight integration of the knowledge-based and data-based functionalities.

This work was subsidized by VLAIO, the Flemish Agency for Innovation and Entrepreneurship and by the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme.

Partial results have been published at the ICLP 2018 [8], RuleML 2019 [26] and DecisionCamp 2019 [24] conferences. We submitted the complete work in this chapter as a journal article [3] at Expert Systems with Applications.

Special thanks go out to my colleague Marjolein Deryck, who co-authored some of the above mentioned papers, based on her experience in a similar use case.

3.1 Introduction

This chapter describes the design, implementation and evaluation of a knowledge base system for the configuration of highly specialized made-to-order products. The products consist of a number of different components, each of which exists in a number of different variants and sizes, and can be produced from different kinds of materials. Customers request a product for a specific set of requirements, encompassing a temperature range under which it should function, pressures it should be able to withstand, size, etc. Typically, the customer is an engineer from another production company. In contrast to typical configuration problems, understanding and explicating the customers' needs is therefore not an issue in this application. The focus of the application is the core design process that the product engineers from the company follow to decide on the design of the product, its specific components and their materials. At the start of our collaboration with the company, the situation was as follows.

Standard Designs Incoming requests were initially received by the sales staff. If the customer's requirements could be fulfilled by one of the company's standard solutions, the sales staff autonomously handled the request. They were supported in this by a Visual Basic tool that inspected a Microsoft Access database to select the appropriate standard design for a particular order. Because of the unstructured way the knowledge in this tool was represented, it was difficult to maintain it. Moreover, this tool only consisted of a small subset of the standard designs, and extending it was a difficult and resource demanding task. The company wanted to move to a more maintainable and expandable solution.

Non-standard Designs Requests that fell outside the standard designs scope were forwarded to the engineering department. Here, one of the engineers analysed the requirements and proposed a suitable product design. If the product was to be used in known areas, characterized by a standard temperature range, a standard product size

range, standard pressure range etc, this was a routine job for the engineer. For these kinds of demands, the company wanted to standardize the design, and automate the process.

If the product fell outside these known areas, the design was more challenging and might require a significant amount of creativity. Although the engineers used computers to perform calculations or create 3D models of the products, there was no software support for the crux of their activity, i.e., the creative design process. The company wanted to provide support for this creative design process for the product engineers.

Design Procedure To design non-standard products, the engineers followed an *ad hoc* process, based on past experience, consultation with colleagues, their own preferences, etc . This way of working is still common in industry. However, it has several downsides. First, the lack of standardization means that different engineers at different locations may come up with different designs for the same set of requirements, some of which may be better than others. Second, the company also depends to a large extent on the expertise of some of its key senior engineers. If they leave the company, a great deal of the knowledge they have built up over the years would leave with them. This would significantly reduce the efficacy of the engineering department. Finally, the lack of software support also means that—in particular, for less challenging design tasks—the engineers spend a substantial amount of time carrying out routine tasks, reducing their efficiency.

Support Tool The company wants to develop a tool to support the design process of non-standard designs. Automated support for (a part of) the design tasks may remediate the downsides of their current way of working. To promote user acceptance and ensure maintainability, the tool should be well understood by the engineers. Over the course of the collaboration with the company, we designed and implemented a knowledge base system that provides a solution to the aforementioned challenges.

A first crucial step is the extraction and representation of the engineers' knowledge of the domain. In this knowledge elicitation process, two parties are involved: domain experts (i.e., the company's engineers) and knowledge engineers (i.e., experts in knowledge representation). Knowledge engineers are not familiar with the problem domain, while the domain experts are not familiar with formal knowledge representation. Proper communication between both parties is therefore very important. Moreover, as the knowledge base aims to reflect a single standard approach, divergent domain expert views need to be aligned. In our project we use the Decision Model and Notation (DMN) [52] to structure and guide the process of capturing and aligning the domain knowledge.

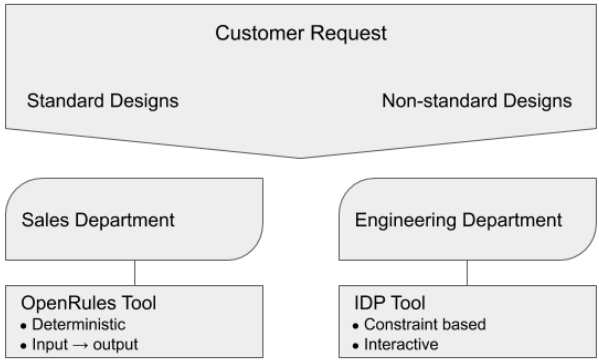


Figure 3.2: Two different approaches for standard and non-standard designs.

A second step is to use the extracted knowledge to build a system that can assist both sales and engineering. However, these two departments have contradictory needs. The sales staff has limited knowledge of product design and expect the tool to propose a single design for each set of input parameters. Moreover, the knowledge in this system should be verifiable by the design engineers to ensure correctness. The engineering team on the other hand, needs the system to assist them in exploring the large and partially unknown design space for non-standard applications. We therefore propose to create two independent (partially overlapping) knowledge bases with different concerns, as can be seen in Figure 3.2. First, we develop a DMN-based tool, implemented in the OpenRules system [53], that helps sales staff select standard designs. This tool has as advantage that the engineers can interpret the knowledge model, which enables them to verify its correctness. Second, to support more challenging design tasks, we develop an interactive interface to consult the knowledge base by means of multiple inferences like propagation, model expansion, relevance checking and explanation. This interface is implemented on top of the IDP Knowledge Base System [14]. In addition to these knowledge-based functionalities, we make use of a number of machine learning techniques to allow the user to explore historically gathered data.

Knowledge in the IDP system is expressed in FO(·), an extension of First Order Logic, which is difficult to understand for domain experts that have no background in logic. We therefore propose to model the same constraint knowledge in the cDMN notation, introduced in Chapter 2.

This research is conducted in collaboration with a company that has engineering and manufacturing offices all over the world. To protect its trade secrets, the company wishes to remain anonymous, and it requested that we avoid providing too much information about its products.

In the remainder of this chapter, we start by discussing the technologies which we will use (Section 3.2), after which a running example will be introduced (Section 3.3). In Section 3.4 and 3.5, we explore a decision procedure and constraint programming approach respectively. Section 3.6 discusses an interactive user interface to support the design process in Section 3.5. Then we investigate how well our cDMN formalism is able to handle the constraint knowledge in this use case in Section 3.7. We conclude by comparing our approach to the literature and a conclusion in Sections 3.8 and 3.9.

3.2 Preliminaries

The engineers' knowledge of the design process will be centralized and formalized in multiple knowledge bases. More specifically, we express decision knowledge in decision tables that follow the DMN standard and implement them in the OpenRules system. Additionally, we also create a knowledge base in the IDP-system. In this section we briefly discuss these technologies in the context of machine component design.

3.2.1 Decision Model and Notation

In [71], the inability to extract knowledge in a format readable by domain expert was identified as a weakness of current product configuration methods. In our project we use DMN to address this concern. This relatively new standard aims to increase the efficiency of decision-making by offering a standard notation for decision logic and "[providing] a common notation that is readily understandable by all business users[...]" [52, p.15]. They claim notation is ideal to discuss decision logic with domain experts.

A detailed description of this notation can be found in Section 2.2.1.

3.2.2 IDP and FO(·)

As we discussed in Section 2.2.2, IDP is an implementation of the knowledge base paradigm. This means that it supports multiple inference algorithms to be executed on the same knowledge specification. Because of this property, IDP is highly suitable to provide multiple functionalities. The functionalities that can be offered are discussed later in Section 3.5.5 and 3.6.

3.3 Running Example

In this section we introduce the running example, which we will use throughout this chapter to demonstrate the key differences between the approaches that are presented. This example is an extract of the complete product design case. Although it is a simplification of the real-world case, it nevertheless retains most of its essential features.

The task of the company's product engineers is to design a product that consists of two mandatory components: a body and a spring. In addition, there may be an additional component added to a design, called a spacer. The engineers are given a number of specifications:

- the requested dimension of the product;
- the temperature range and pressure in which it should function;
- whether there is a risk of pressure from the back side of the product (back pressure);
- the position in which the product will be installed.

They need to determine:

- Which type of design (closed or open) to use;
- Whether to use a standard or thicker spring and which material to use for it;
- Which material to use for the body;
- Whether to use a spacer and, if so, which material to use for it;

They have to make these choices in such a way that:

- The design can release built-up pressure if necessary
- The materials used can cope with the given temperature range;
- When the materials shrink due to cold, the spring should prevent the component from falling out of the cavity in which it is placed;

The materials have different costs, with cheaper materials typically being weaker than more expensive ones. In general, the engineers look for the cheapest design that will work.

3.4 Modelling the Design Process

The engineers have a "standard" decision process that they use to handle routine requests. However, this process is not explicitly standardized and different engineers at different locations may do certain things somewhat differently. To fully standardize this process and to be able to automate it, the engineers' detailed technical knowledge needs to be represented in a formal and structured manner. In this section we describe the substantial knowledge elicitation effort needed to gain insight in the tacit design knowledge of different engineers, and to unify and formalize them in decision models.

3.4.1 Knowledge Extraction of the Design Process

As we described in Section 3.2.1, the ability of DMN to be understood by "business users" (in this case, the product engineers) is a key advantage. It established the common language that could be understood by both our knowledge engineers and the company's domain experts. This helps to avoid misunderstandings, ensures smooth communication, and allows certain well-delineated parts of the decision model to be assigned as "homework" to specific experts. Moreover, the product engineers' ability to not only understand but also help to build a formal model is of significant importance for the maintenance of the decision model, for which the engineers themselves will be responsible.

To create a DMN model, we chose to start from a series of brainstorming workshops with all the involved parties. Each workshop takes a couple of days and results in an initial representation of the design process for a specifically delineated application area. The involved parties are a number of design engineers (representing each of the locations worldwide that are involved in the particular application area), a manager and one external knowledge engineer to guide the workshop. This approach offers a number of advantages.

- Since multiple participants are involved, we do not blindly adopt the approach of one engineer or one particular location.
- The face-to-face time allows intensive discussion about why certain decisions are taken, which is often necessary when different engineers are used to following different approaches.
- During one multiple-day workshop, all parties focus solely on one specific application, which helps to keep the discussion focused.
- The knowledge engineer not only helps with technical issues concerning the representation, but he also assists the engineers in clarifying their design process:

as a non-expert in the domain, he is able to ask "trivial" questions that help to ensure that all the engineers are on the same page and that nothing is being overlooked.

Each workshop starts with a brief introduction to DMN, after which the knowledge engineer guides the domain experts through the modelling process. We typically start by constructing a Decision Requirement Diagram (DRD), such as the one in Figure 3.3, to get a general overview of the structure of the design process, and then proceed to construct detailed decision tables for each of the decisions.

A workshop results in a formal representation of the engineers' relevant knowledge, which is then used to build an initial prototype of a decision support system for that particular application area. This prototype is then presented to the design engineers for evaluation. The evaluation can be done briefly by e-mail or in another workshop, depending on how close to reality the preliminary model is. Based on the feedback, the model is refined. This process is repeated until all parties agree that the model is correct.

Results

Following the methodology outlined in Section 3.4.1, we have extracted the knowledge of the routine design process in six different application fields, producing a total of approximately 190 decision tables. In each of the applications, one or two tables were pure data tables, consisting of numerical data for dimensioning the components. Since the discussed applications are more or less similar, some of the already constructed data and decision tables from one application could be reused in another. The size of the extracted tables ranges from 1 to 38 rows, with an average size of 5 rows and 3 input conditions.

A small exception to our normal way of working occurred when representing the design process for applications that fall within the scope of the Visual Basic tool that had already been developed for the sales staff. Here, we simply started from the existing an VB code and transformed this into a DMN model, which proved to be significantly shorter (over 200 database rows and 360 lines of corresponding VB code were reduced to 80 table rows spread across 20 tables), more structured, and therefore easier to maintain.

Overall, the DMN representation seemed to fit well with the company's engineers' own way of thinking about their design process. They found the DMN format quite intuitive and after some initial questions, they were typically able to easily interpret and reason with the knowledge in the tables. Our experiences therefore confirm that DMN's readability for domain experts is a big advantage of this standard. However, there were some exceptions. Occasionally, the engineers do not follow a strict bottom-up decision

procedure when making their design. For instance, sometimes a thicker spring or closed design type needs to be used to ensure that the component stays in place. Selecting a thicker spring and keeping an open design type is the preferred approach, but this is not always feasible. In particular, in cold circumstances, the product may shrink to such extent that the open design would fail, even when using a thicker spring. However, to know whether this is the case, the shrinkage of the product has to be computed. Because this shrinkage depends on the materials being used and the precise layout of the components, this computation can only be done at the very end of the design process. Therefore, currently the engineers assume that the option with the open design type will work. They completely design the product based on this assumption and then compute the shrinkage. If it turns out that the shrinkage is too big, the engineer restarts the design process with a closed design. The knowledge needed to implement such a "guess and check" procedure cannot be elegantly represented in DMN. In Section 3.4.2 we discuss the work-around that we have used for this.

In general, we perceived the use of a formal representation in the workshop as a significant added value. The precision of the notation allowed us to quickly detect inconsistencies and missing values in the information that the domain experts were providing. Moreover, the design engineers themselves also started to notice flaws in the decision tables once they had gotten used to the notation. For example, they noticed implementation mistakes made by the knowledge engineer, and thought of previously unnoticed exceptions to their own design process. Towards the end of a workshop, the design engineers were comfortable enough with the notation to leave certain decision tables to be constructed as "homework" after the end of the workshop.

Based on these experiences, we are confident the design engineers will be able to maintain the existing decision tables and will be able to construct additional DMN models for new application areas.

3.4.2 DMN Model of Running Example

In this section, we present a DMN model that is the result of the DMN workshops held with the product engineers. As we will show, some parts of their design procedure fit quite naturally into the DMN framework, while others can only be encoded by means of rather cumbersome work-arounds.

Figure 3.3 shows the DRD of our model, which captures the general structure of the decision logic. It starts from the input customer requirements (product dimensions, operating conditions) at the bottom and has the decisions that must be made at the top, with certain sub-decisions in between. The ellipses represent inputs, whereas each rectangle corresponds to a decision table that can be found in Figure 3.4, 3.5, 3.6 and 3.7.

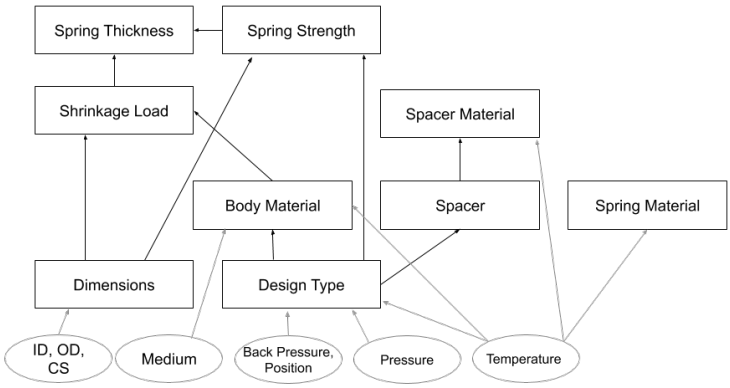


Figure 3.3: DRD of the product design application.

Dimensions						
U	**OD**	**ID**	**CS**	**OD**	**ID**	**CS**
1	null	–	–	ID+2*CS		
2	–	null	–		OD-2*CS	
3	–	–	null			(OD-ID)/2

Figure 3.4: Decision table for dimensions.

We now discuss the different steps and DMN representation of the design procedure that the engineers typically follow.

1. There are three relevant *dimensions*: the outer diameter (OD) of the product, its inner diameter (ID) and its cross-section (CS). The customer provides two of these, and the engineer computes the third, using the formula $CS = (OD - ID)/2$.

   The corresponding DMN table in Figure 3.4 consists of three different rows (one per dimension that might be missing), that all contain essentially the same information.

2. An initial *design type* is chosen, depending on the required operating conditions, whether there is back pressure and in which position the product will be installed.

   Even though this decision fits well within the mould of a DMN decision table, as can be seen in Figure 3.5a, it is not clear what the underlying reasons behind this table are.

   We will discuss this in detail in Section 3.4.4.

3. A *spacer* is selected based on the design type, in the second decision table in Figure 3.5b.

Design Type					
A	**Back Pressure**	**Position**	**Pressure**	**Temperature**	**Design Type**
1	Yes	Pressure Accumulating	–	–	Open
2		Bi-Directional	–	–	Open
3		not(Pressure Accumulating, Bi-Directional)	> 150	–	Open
4			(100, 150]	≥ -50	Open
5			(100, 150]	< -50	Closed
6			≤ 100	–	Closed
7	No	–	–	–	Closed

(a) Design Type

Spacer		
U	**Design Type**	**Spacer**
1	Closed	No
2	Open	Yes

(b) Spacer

Figure 3.5: Decision tables for Design Type and Spacer.

Body Material			
P	**Temperature**	**Design Type**	**Body Material**
	[-150.. 200]	*Closed, Open*	M_2, M_1, M_3
1	[-150, -120)	–	M_1
2	[-120, 80]	Closed	M_1
3	[-120, 80]	Open	M_2
4	(80, 100]	–	M_1
5	(100,200]	–	M_3

(a) Body Material

Spring Material		
U	**Temperature**	**Spring Material**
1	[-150, -50)	M_1
2	[-50,150]	M_5
3	(150,200]	M_3

(b) Spring Material

Spacer Material			
U	**Spacer**	**Temperature**	**Spacer Material**
1	Yes	[-150, -50)	M_1
2		[-50,150]	M_5
3		(150,200]	M_3
4	No	–	null

(c) Spacer Material

Figure 3.6: Decision tables to determine component materials.

4. Based on the temperature restrictions and design type, the best *material* is selected for the body, spring and spacer, as can be seen in Figure 3.6.

Spring Strength				
U	**Design Type**	**CS**	**Standard Spring Strength**	**Thick Spring Strength**
1	Closed	[0.75, 1.25]	15	20
2	Closed	[1.75, 2.25]	13	18
3	Open	[0.75, 1.25]	7	11
4	Open	[1.75, 2.25]	6	9

(a) Decision table for Standard Spring Strength and Thick Spring Strength

Shrinkage Load		
U	**Design Type**	**Shrinkage Load**
1	Open	f(CS, Material of Body, Temperature)
2	Closed	f(CS, Material of Body, Temperature)

(b) Calculation of Shrinkage Load

Spring Thickness		
U	**Shrinkage Load**	**Spring Thickness**
1	$\leq$ Standard Spring Strength	Standard
2	(Standard Spring Strength,Thick Spring Strength]	Thick

(c) Decision table for the Spring Thickness

Figure 3.7: Decision tables to check whether a design can cope with the expected shrinkage.

5. The expected *shrinkage* is computed based on the design type, dimensions, temperature and selected materials.

   While we omit the details, the formula for this calculation can easily be placed in the "Shrinkage Load" table in Figure 3.7b.

6. Based on the selected design type, the *spring strength* of a standard spring is determined. If this is enough to cope with the expected shrinkage, the standard spring is selected. Otherwise, the engineers switch to a thicker spring and recompute the spring strength of the design.

   Since DMN does not allow to recompute a value, the "Spring Strength" in Figure 3.7a table computes both the spring strength that the design would have with a standard spring and that it would have with a thicker spring. The "Spring Thickness" table in Figure 3.7c then uses these two values to decide which spring type to use. Because of this, the DRD has an edge *from* "Spring Strength" *to* "Spring Thickness". This is counter-intuitive, because in reality it is the thickness of the spring that determines the design's spring strength, not the other way around.

7. If the thicker spring from the previous step does not provide enough spring strength, the engineers' final option is to switch to another design type. Since the choice of design type is the initial choice upon which all further choices are

based, this means that the engineers essentially restart the entire process from scratch.

When no acceptable spring can be found, the DMN table “Spring Thickness” returns `null`. This alerts the users of the system that something is wrong. However, in DMN, it is not possible to express that, in this scenario, the initial choice for “Design Type” was incorrect and should be remade to select another design type. Therefore, we have implemented a workaround in which the “Design Type” table is overly cautious: in low temperatures, it will always choose the more shrinkage resistant closed design. This means that the product will be over-designed in certain circumstances, but it prevents possible failure of the design process.

3.4.3 Results

DMN is designed to be a fully executable specification and is currently supported by a number of different tools, both commercial and open source. We have created an automated design system by implementing the constructed DMN tables in the OpenRules [53] system.

Starting from a direct formalization of the engineers’ design process proved noticeably more useful. Not only did the engineers appreciate the intuitive way of reasoning in the DMN standard, it also made them think about how they come to a design in a given situation. It also helped them to better understand and standardize their design process. Moreover, when transforming the Visual Basic tool developed for the sales staff into a DMN model, a number of irregularities surfaced. Without a formal representation of the knowledge, it would have been far more difficult and time consuming to detect these faults. Indeed, these faults were not detected during the multiple months of using the Visual Basic tool, whereas they were easily identified once the knowledge was formalised in DMN.

To analyse the potential of the DMN-based approach, the engineers inspected the decision tables in detail and provided us with ten cases that represent both normal sets of requirements and a number of edge cases. Our OpenRules implementation using the DMN model generates the correct design in all of the test cases. Computing a design takes about 0.3 seconds on one core of an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.

Based on these promising results, they decided to adopt this approach. They further developed the OpenRules models and subjected them to extensive testing. Currently, the DMN-based approach is operational in their production environment for a select number of application areas.

3.4.4 Discussion

The direct encoding of the design engineers' design process has the advantage that it is easy to implement and to understand. However, there are also downsides to this approach.

First, as mentioned in Section 3.4.2, a few aspects of the design process do not fit readily into the DMN model. Currently, we have worked around this problem by an "err on the side of safety" approach. While the solution may be suboptimal (in the sense that the product is over-designed), it prevents the risk of suggesting faulty designs, and avoids the introduction of complicated decision structures that would reduce the legibility of the DMN model.

Second, because decisions in DMN are made sequentially, as denoted in the DRD, it can be difficult to model interwoven decisions. The selection of the design type and spacer is an example of this. An open design and a spacer always occur together. While some circumstances require a closed design, other circumstances require the use of a spacer. Because it is obligatory to specify a predefined decision order, the rationale for the use of a spacer and the design type should be put within a first decision table, while its outcome can be used as an input for a second decision table. Choosing which is the first and which is the second decision is arbitrary.

Third, the DMN representation forces one to mix different kinds of knowledge within a single table. For instance, the decision table "Design Type" in Figure 3.5a is based on both physical constraints and preferences of the company. However, the actual constraints and preferences cannot be deduced from this table alone. For instance, the decisions could be explained in any of the following ways:

- A closed design is always preferred, but it is unusable in situations 1, 2 and 3;
- An open design is always preferred, but it can only be used in situations 1, 2 and 3;
- An open design is preferred if there is back pressure, while closed designs are preferred in all situations when there is no back pressure.

Mixing constraints and preferences this way reduces the maintainability of the tables. Suppose that a supplier changes the price of some material. This may have an impact on the preferred shape of design, as the engineers try to design the product at the lowest cost and closed designs require more material. However, without knowing the underlying rationale of this table, it is impossible to judge the impact of this change on the decision tables in Figure 3.5. A representation that separates preferences from constraints would not have this problem.

Fourth, the available DMN inference engines support only a single inference task, namely that of computing the "output" decision variables given all the input variables. However, in a system that is used interactively by a design engineer, we can also envisage other useful inference tasks. For instance, after filling out only a subset of the input variables, the engineer may be interested in knowing whether a particular design is still possible. Or, in discussions with a customer, he may be interested in knowing which values of the input variables would allow the use of a thinner spring.

Fifth, DMN keeps the complexity of the decision process manageable by splitting it into different decision tables. A downside of this approach is that it is not possible to take into account global properties of the design. For instance, we may be interested in selecting the cheapest possible design. The cost of a design depends on which parts are included in the design and on which materials are used to make these parts. These decisions influence each other: certain parts can only be made of certain materials, while the use of a better material might eliminate the need for a particular additional part. This interdependency means that we cannot aspire to consistently find the cheapest global design by making a sequential series of local decisions.[1]

Finally, the entire DMN approach of course assumes that there *is* a decision procedure to model. For the design of products in new and challenging application areas, the design process is unknown. This means that DMN cannot be used at all, even though (some of) the information from known areas may still be applicable.

We conclude that a DMN model of the engineers' design procedure is well-suited for standard applications. To support the engineers with non-standard applications, however, a different approach is needed.

3.5 Modelling the Underlying Constraints

As discussed in the previous section, we cannot hope to achieve all of our stated goals by an approach in which we simply use a direct implementation of the design procedure as the engineers follow it. We need to clarify the preferences and the underlying physical constraints that have led the engineers to adopt this procedure in the first place.

In general, the design process followed by the engineers is governed by a number of physical constraints (e.g., a material M_1 can only be used in temperatures $\leq 100°C$) and preferences (e.g., material M_2 is preferred over material M_1, perhaps because it is cheaper or more durable). In order to develop a decision support system that can also provide useful information for challenging new application areas, we need to make direct use of these underlying constraints and preferences, rather than of the engineers'

[1] A limited form of local (i.e., within the scope of a single decision table) optimization can be done using the "Priority" hit policy

existing design process. These constraints provide more information than is explicitly present in the design procedure, because they also explain *why* certain designs are impossible. As such, it is not possible to automatically deduce these constraints from the design procedure. Instead, coming up with them requires additional discussions with the design engineers.

3.5.1 Knowledge Extraction of the Physical Constraints

Although some information on the physical constraints of components and materials is documented, the majority of this information exists only in the engineers' head. The elicitation of this information proved difficult. The engineers often did not know where to start and discussions tended to be chaotic. To remediate this, we used the DMN models to structure the discussion.We organized a discussion with the engineers who were originally involved in the construction of these models. For each row of each table, we asked *why* this output is the appropriate choice in this particular set of circumstances. Unlike the workshops in which the DMN models were initially constructed, here it is less crucial to involve different engineers: even though different engineers may disagree on the best solution for a given problem, they tend to agree on the reason why certain solutions might or might not work.

This use of the DMN tables provides a structured way of working, in which different topics are addressed in a meaningful order and we can be sure that no relevant constraints will be overlooked. Moreover, because the engineers know and understand the DMN model, there is no confusion about the question that is being discussed at a particular point in time.

To reduce the time investment required from the engineers, it proved useful to carefully prepare these discussions in advance. Often, the form in which a particular table has been written down already suggests a certain underlying reason. For example, the "default" row at the bottom of the decision table "Design type" in Figure 3.5a suggests that the closed design is the preferred choice, with the other rows describing circumstances in which this preferred choice is not possible. Additionally, considerations that were mentioned during the workshops to construct the DMN models may provide further clues. In practice, we have found that we can construct most of the constraints without help of the engineers and only need them to verify and help us revise our initial guesses.

From a theoretical point of view, it is interesting to notice that, although the physical constraints have a similar structure, two kinds can be discerned. Some of the constraints are of descriptive nature, e.g., only open designs can release pressure. Other are prescriptive in nature, e.g.; if the product is used in a pressure accumulating location, then it should be able to release pressure.

1. $DesignType =$ "$Open$" $\Leftrightarrow$ $ReleasePressure$.

2. $Location =$ "$PressureAccumulating$" $\Rightarrow$ $ReleasePressure$.

Both kinds are used next to each other.The practical relevance for our current application seems limited, but this might become more important when technology evolves.

Even though most of the DMN decision tables can be interpreted individually, in some cases, multiple decision tables are interwoven. As a result, the constraints that need to be added to model these decision tables might focus on the combination of multiple tables. Section 3.5.3 handles a detailed example of this.

3.5.2 Knowledge Extraction of the Preferences

Typically, there will be many different designs that are physically feasible and therefore satisfy all of the constraints. The engineers have to select the *best* design among the possibilities. In theory, a large range of optimization criteria and techniques could be used for this. In practice however, these are limited by the available data, incomplete knowledge of client preferences, and so on. Therefore the purpose of preference elicitation is the creation of a flexible optimization framework that includes multiple criteria, and which can be adapted to client needs. The available metrics can be grouped around different criteria, such as cost and durability. In our case we have both totally ordered metrics and partially ordered ones. Some orderings have a fixed, meaningful distance, but this is not the case for all orders. These differences between the optimization metrics need to be taken into account when combining them.

One of the explicit goals of our project is to help the company increase its internal standardization and reduce the impact of preferences of individual engineers. At the same time, however, the engineers should retain enough flexibility to take the wishes of individual customers into account. Our optimization framework uses a three-level structure to this end. The highest level consists of a lexicographic ordering of the solution space, according to a sequence of criteria $m_1, \ldots, m_n$. This means that all solutions that have a non-optimal value for m_1 are immediately discarded. Among the remaining solutions, those with a non-optimal value for m_2 are also discarded, and so on. The first criteria $m_1, \ldots, m_{n-2}$ are all atomic criteria, such as the local availability of materials.

To increase confidence in the system, we want to be sure that no design exists that is better than the proposed solution in any of the criteria. The criterion m_{n-1} is the second level of our hierarchy and calculates the pareto-optimal front of the remaining solutions with regards to a new set of metrics $m'_1, \ldots, m'_k$. Such a pareto-optimal front is a set of solutions that have the property that no single criterion m'_n can be improved

without worsening another criterion. The result is a collection of possible solutions with different characteristics.

At the third level, i.e., the final criterion m_n, the engineer can select the best design according to his preferences. With a slider the engineer may indicate the importance of a limited number of numerical of criteria, such as cost and life expectancy, for the final solution. This is translated into a weight for the corresponding term in a weighted cost function.

$$lexico\big(m_1, \ldots, m_n, pareto(m'_1, \ldots, m'_k), max(\sum_i \alpha_i m''_i)\big) \tag{3.1}$$

where

m_i and m'_i are partial orders

m''_i are numerical criteria

the α_i are weights to fine-tune the importance of each of the numerical criteria

This framework reconciles the company's need for standardization with the product engineers' need for flexibility. It does so by confining the solution space and by controlling the optimization metrics on the one hand and by interactively adjusting the importance of metrics to reflect clients' needs on the other hand.

3.5.3 IDP Model of Running Example

In this section we apply the constraint and preference approach to the running example described in Section 3.3. We use the FO(·) language to represent them. We explain how the formulated constraints relate to the DMN model that we presented in Section 3.4.2.

Dimensions A constraint such as $CS = (OD - ID)/2$ expresses a relation between three variables. Unlike in DMN, there is no need to designate some of these variables as input and the others as output. Therefore, there is no need to produce three separate table rows as in Figure 3.4: the single constraint can be used to derive the value of the remaining variable from that of the other two.

Design Type and Spacer The restrictions on *Design Type* and whether a *spacer* is used are interwoven and more complex. Discussions with the engineers have revealed that the decision tables in Figure 3.5 can be explained as a combination of the following constraints:

1. Only open designs are able to release pressure.
   $DesignType = \text{“}Open\text{”} \Leftrightarrow ReleasePressure.$
2. When the product is placed in a pressure accumulating location, it should be able to release pressure.
   $Location = \text{“}PressureAccumulating\text{”} \Rightarrow ReleasePressure.$
3. It is impossible to use a spacer in combination with a closed design.
   $\neg\big(DesignType = \text{“}Closed\text{”} \wedge ComponentUsed(Spacer)\big).$
4. A spacer is necessary if the back pressure is higher than 150 bar.
   $BackPressure \wedge Pressure > 150 \Rightarrow ComponentUsed(Spacer).$
5. In the bi-directional location, the component tends to move back and forth excessively, so in order to avoid damage, a spacer is always necessary.
   $Location = \text{“}Bi\text{-}directional\text{”} \Rightarrow ComponentUsed(Spacer).$
6. Each design should always have a body and a spring.
   $ComponentUsed(Body).\ ComponentUsed(Spring).$

And the following preferences:

1. If the back pressure is between 100 and 150 bars, it is recommended but not mandatory to use a design with a spacer.
2. Closed designs tend to be cheaper and outperform open designs, so they are the preferred type of design.

The first line in table “Design Type” in Figure 3.5a is a result of combining constraint 1 and 2. The component should be able to release pressure and since closed designs cannot do that, an open design is the only option.

The second row is a combination of constraint 3 and constraint 5. In the “Bi-directional” location a spacer is always necessary, and since it is impossible to have a spacer in closed designs, the only remaining possibility is to go for an open design.

Analogously, the third line in the decision procedure can be obtained from combining constraint 4 and 3.

The combination of constraint 3 and preference 1 would suggest that an open design would be selected if the back pressure is between 100 and 150 bars. However, in the decision table, a distinction is made between row 4 and 5 based on the temperature. This is the previously mentioned workaround to avoid running into cases in which the shrinkage cannot be addressed by simply adding a thicker spring to an open design. By contrast, our constraint representation avoids the need for such a work-around (see below).

In all other situations, both closed and open designs are possible, but closed designs are preferred, which explains the last row in the decision procedure.

Materials If a component is used, it should have a material; a component's material is *null* when it is not used. Each material has a minimum and maximum temperature. For each component that is used in the design, a material must be selected such that the operating temperature falls within the temperature range of this material. This knowledge is represented by the following two constraints:

$$\forall c[Component] : ComponentUsed(c) \Leftrightarrow Material(c) \neq null.$$

$$\forall c[Component] : ComponentUsed(c) \Rightarrow$$

$$MinTemp(Material(c)) \leq Temperature \leq MaxTemp(Material(c)).$$

The specific values for $MaxTemp$ and $MinTemp$ of each material are predefined, and can be found in Figure 3.17c.

Moreover, not every material can be used for each component, i.e., only a subset of all possible materials can be used. This is expressed in the predefined *PossibleMaterial* predicate:

$$\forall c[Component] : ComponentUsed(c) \Rightarrow PossibleMaterial(c, Material(c)).$$

In addition to these general constraints on the materials of all components, there is also a specific constraint on the material used for the body:

$$DesignType = Closed \Rightarrow Material(Body) \neq M_2.$$

These constraints still leave considerable freedom in selecting a material. In our original DMN model (Figure 3.6), we selected a specific material for each set of circumstances, based on the two physical constraints, and on the fact that the engineers prefer to use cheaper materials when possible. With constraint-based reasoning, we can separate these two concerns, expressing the constraints in the tables mentioned above, and subsequently expressing an optimization criterion as discussed in Section 3.5.3.

To illustrate how these constraints and preferences correspond to the "Body Material" table in Figure 3.6a, we discuss one row in this decision table. Based on the constraints and preferences, row 5 can be explained as follows: in this temperature range, all three materials M_1, M_2 and M_3 are possible; however, if a closed design would be selected, material M_2 can no longer be used. From the remaining materials (M_1 and M_3), the cheapest (M_1) is then selected. This is indeed the outcome of the decision table. All other rows in the table can be explained analogously.

Shrinkage check The $SpringStrength$ should be larger than $ShrinkageLoad$:

$$ShrinkageLoad < SpringStrength. \tag{3.2}$$

Both $ShrinkageLoad$ and $SpringStrength$ can be calculated in terms of previously discussed design features, such as *Design Type*, *Cross-Section* and *Spring Thickness*. The IDP-system ensures that this shrinkage check is satisfied by adjusting the design features, for example, by selecting a thicker spring or a different design type.

Preferences

In the current version of the application, the optimization framework described in Section 3.5.2 is implemented as a weighted cost function. Lexicographical criteria are modelled as n-category integers $[1, n]$, and the corresponding weight is set to a sufficiently high value to ensure that no combination of other criteria can compensate for a non-optimal value. Another concern is to ensure that only pareto optimal solutions are proposed, even for categorical criteria. This is implemented by introducing ascending numerical values for those categorical criteria. By ensuring all pareto criteria are included in the weighted sum, only pareto optimal solutions will be proposed.

There are four criteria that are relevant to select an optimal design.

- The *cost* of a design can be determined by adding up the cost of each of the used materials, listed in Figure 3.19. Note that a design without a spacer will be cheaper, since it requires less material. Moreover, thicker springs are harder to fabricate, which makes production cost higher.
- The *durability* of a design depends on a number of factors. When a product without spacer is used in a back pressure environment, that product will wear faster. When this pressure is higher than 100 bar, the product is damaged almost immediately. Moreover, a product wears because of the spring strength.
- The *leak tightness* of a product is proportional to the spring strength. For example, closed designs typically have a higher leak tightness than open designs.
- *Availability*: certain materials could be out of stock and should be avoided if possible. This is the most important criterion in the lexicographical ordering.

In the framework proposed in Section 3.5.2, these criteria should be combined as $Lexico(Availability, Pareto(Durability, LeakTightness), Min(Cost))$. In our current implementation, we approximate this framework by a custom weighted sum function. These weights can be tweaked to favour one criterion over the others, for example, when setting the weight of the cost at 5, the cost will be a more dominant criterion.

The following extract shows the logic to construct the optimization term: Here, we use the FO(·) notation of $sum\{\bar{x} : \varphi(\bar{x}) : F(\bar{x})\}$ to denote $\sum F(\bar{x})\ \ s.t.\ \ \varphi(\bar{x})$.

$$
\begin{aligned}
&Cost\ = \\
&\quad sum\{c : IsUsed(c) : Cost(Material(c))\}+ \\
&\quad sum\{\ : SpringThickness = Thick : 5\}+ \\
&\quad sum\{\ : SpringThickness = Standard : 2\}. \\
&Durability = \\
&\quad sum\{\ : \neg IsUsed(Spacer) \wedge BackPressure \wedge Pressure < 100 : -5\}+ \\
&\quad sum\{\ : \neg IsUsed(Spacer) \wedge BackPressure \wedge Pressure \geq 100 : -100\}+ \\
&\quad -1/5 \times SpringStrength. \\
&LeakTightness\ =\ 1/2 \times SpringStrength. \\
&Availability = \\
&\quad sum\{c : IsUsed(c) \wedge \neg LocallyAvailable(Material(Component)) : -100\} \\
&OptimizationFunction\ = \\
&\quad Durability + LeakTightness - 5 \times Cost + Availability.
\end{aligned}
$$

By capturing all optimization criteria in one optimization function, we can make use of the IDP *minimization* inference task to find an optimal solution. We allow the user to adjust these weights at execution time. Hence, engineers can choose which criterion should be given priority within this preference framework

The combination of constraints and preferences defined in this section produce the same outcome as the DMN procedure of Section 3.4.2. For example, an open design will only be used when a spacer is not necessary (because it is less leak-tight), cheaper materials are selected when they can handle the operating conditions, and so on. A more detailed comparison is made in Section 3.7.6.

Separating the preferences and constraints does have a number of flexibility advantages.

- The optimization criteria can be easily adjusted by tweaking the weights, depending on the customer's priorities.

- When a material cost or local availability changes, this can easily be updated in the structure. There is no need to change the constraints or preferences.
- When adding an additional criterion this can be done in a modular way.

This way of working stands in sharp contrast with how changes manifest themselves through DMN models. Because the different concerns are mixed in a single decision table, adding or changing a certain preference (e.g., the cost of a material) might require to revise the complete DMN table.

3.5.4 Missing Data

In a different application area, we encountered an additional challenge. When enumerating the design properties and material properties, it turned out that certain data points were unknown, e.g., the property $LifeExpectancy$ of a design is known for a number of designs, but missing for others. The company's engineers indicate that this is indeed the current situation. It would take great effort to determine these missing values and complete the data in the property tables. Therefore, a solution should be developed that allows our system to handle such missing data.

We introduce a small example to illustrate the mechanism we propose to cope with missing data. Each of the properties that have missing values, becomes a partial instead of a total function, e.g., $partial\ LifeExpectancy(Design) : type$. The structure only defines the values of this function that are known, e.g., $LifeExpectancy = \{D1 \mapsto x; D3 \mapsto z\}$, when the $LifeExpectancy$ of design $D2$ is unknown. Then, the constraint states that $LifeExpectancy(UsedDesign) > RequestedLifeExpectancy$ is changed to one of the following logical formulas:

$$\forall x : LifeExpectancy(UsedDesign) = x \Rightarrow x \geq RequestedLifeExpectancy.$$

$$\exists x : LifeExpectancy(UsedDesign) = x \wedge x \geq RequestedLifeExpectancy.$$

The first constraint specifies that if the $LifeExpectancy$ is known, it should satisfy the constraint. This approach is brave when it comes to accepting designs with missing values, i.e., when a value is missing, the $LifeExpectancy$ constraint is satisfied, and the corresponding design can be selected. The second constraint is more cautious. It only allows designs which have no missing values, by stating that a value should exist for the $LifeExpectancy$ of the $UsedDesign$ and this value is higher than the requested one.

The engineers could not specify a general desired behaviour of the system, since this greatly depends on the situation. They would like to be able to switch between those

two behaviours manually. Therefore we slightly adapted our approach to give them this choice. We introduced a new predicate in the vocabulary: $AllowUnknown(Property)$; the constraint is changed to:

$$\big(\exists x : LifeExpectancy(UsedDesign) = x \wedge x \geq RequestedLifeExpectancy\big) \vee$$

$$\big(\neg\exists x : LifeExpectancy(UsedDesign) = x \wedge AllowUnknown(LifeExpectancy)\big).$$

If $\neg AllowUnknown(LifeExpectancy)$, designs with an unknown $LifeExpectancy$ are not considered and vice versa. This enables the user to specify whether to consider unknown values for each design and material $Property$.

3.5.5 Implementation of a Constraint-based Approach

We have used the IDP knowledge base system [14] to implement the constraint-based design system. It offers different algorithms, implementing a number of logical inference tasks, based on Answer Set Programming (ASP), Logic Programming (LP) and SAT solving technology. In previous editions of the ASP Competition [10], IDP proves to be competitive with other state-of-the-art ASP systems, though typically somewhat slower than systems such as Clasp.

Our main reasons to use IDP are twofold. First, IDP uses classical logic as an input language. Individual constraints are represented independently of each other. Thanks to this modularity, they can be explained to the company's design engineers without requiring much additional background. While the engineers would probably not be able to write down constraints correctly, they are able to read them pretty well. We suspect that this would not be the case for languages such as ASP, that use non-classical connectives such as negation-as-failure.

Second, as IDP is an implementation of the knowledge base paradigm, it provides support for different logic inference tasks on the same knowledge base. This can be used to provide a multitude of functionalities to the end user. Such flexibility is only partially offered by constraint-programming languages such as MiniZinc [50].

To reproduce the functionality of our DMN implementation in OpenRules with our constraint-based representation in IDP, we make use of the inference task of *Model Expansion* and *optimization*. Other inference tasks are used in our GUI, which we discuss in Section 3.6.

Model Expansion To compute a design, we apply the logical inference task of Model Expansion [49]. This takes as input a theory T and a structure S_{in}, i.e., an

interpretation to the types, predicates and functions of a part of the vocabulary of T. The goal is to produce a structure S_{out} for the remaining part of the vocabulary such that $S_{in} \cup S_{out} \models T$. In our case, the structure S_{in} describes the problem specification, by providing an interpretation for constants such as $Temperature$, $Pressure$ and $Location$; the structure S_{out} then describes a design, by providing an interpretation for constants such as $DesignType$ and functions such as $Material$, which maps each component used in the design to the material it should be made from.

Optimization Whereas IDP's model expansion functionality finds any model satisfying (T, S_{in}), its optimization functionality allows us to specify an additional numerical term t. IDP then computes not just any solution to the model expansion problem, but the solution S_{out} that, in addition to being such that $S_{in} \cup S_{out} \models T$, also minimizes the value $t^{S_{in} \cup S_{out}}$ of this term. In our case the term is the weighted sum $OptimizationFunction$, as defined in Section 3.5.3. IDP implements this inference task with a combination of techniques such as exploiting symmetries, functional dependencies, symbolic unit propagation and an optimization loop, which iteratively produces better solutions by each time adding a new constraint that the next solution must have a lower score than the previous solution. This last technique is typically also used in, e.g., ASP solvers.

3.5.6 Results

While it proved relatively easy to construct the DMN model in collaboration with the engineers, constructing the more expressive FO constraint-based model was significantly more challenging. We therefore used the former to validate the latter. In particular, we check the correspondence between the output $D(I)$ of the DMN model D for a given input I and the solutions S_{out} of the model expansion problem (T, S_I) for the FO theory T. The vocabulary of the theory T was chosen such that the DMN input I and output $D(I)$ can easily be translated into structures S_I and $S_{D(I)}$.

The first aspect to check is that the constraints are not too strict: for each possible set of inputs I, the design $D(I)$ that would be constructed by the DMN model should satisfy the constraints in theory T, i.e., $S_I \cup S_{D(I)} \models T$ or in other words, $S_{D(I)}$ should be a solution the model expansion problem (T, S_I).

Second, to verify that the constraints are not too weak, we also checked that the design $D(I)$ proposed by the DMN model D is among the optimal solutions of this model expansion problem, i.e., that $t^{S_I \cup S_{D(I)}} \leq t^{S_I \cup S'}$ for any other solution S' to the model expansion problem (T, S_I), where t is the optimization term that should be minimized. This both checks that the constraints do not fail to rule out designs with a higher score

that are in fact impossible and that the weights used in the optimization criterion are assigned correctly.

We implemented both of these checks using IDP. We first transformed the DMN model to IDP syntax as described in [20]. We then used IDP to perform the required checks on the relation between the IDP theory derived directly from the DMN model and the IDP theory that represents the constraints.

The first check initially revealed a small number of errors in the constraint-based representation. After minor fixes it was concluded successfully. The second check then revealed that, in a number of cases, the constraint-based model produced more optimal designs than the DMN model. While we initially thought that this was due to more errors in the constraints, an analysis together with the design engineers revealed that the outcome of the constraint-based model was in fact correct and that their own design process was in these cases non-optimal. This non-optimality turned out to be caused by the difficulty of making the decisions in a predetermined order. When using the constraint-based method, on the other hand, it is not necessary to determine such a decision order to find a better scoring global optimum.

The IDP system typically finds the optimal design in about 3.15 seconds on one core of an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.

3.5.7 Discussion

The constraint representation has a number of interesting advantages. By modelling the required characteristics of the product as constraints rather than specific rules, the outcome is usually a set of solutions rather than a single solution. Hence, the constraint approach enables flexible optimization, allowing a solution that is optimal in a specific situation to be chosen from this set of solutions. Moreover, it allows distinguish between physical constraints and subjective preferences. This in turn, has the advantage that changes in the underlying information can easily be processed in the model without the need to review the entire decision process. Pieces of information might also be reused in new application areas for which no complete model is available.

However, we also found a disadvantage of the approach. The constraint approach is less comprehensible for domain experts. On the one hand, the syntax for writing down individual constraints is more complex. While we have used FO($\cdot$) because we believe it is quite understandable for untrained domain experts, it is still more complex than the simple table-based DMN format. On the other hand, reasoning with constraints is inherently more complex than the straightforward assignment of values in the DMN approach. There is always a clear link between input and output in a DMN decision model, which makes it easy to interpret and inspect by a domain expert. When using constraints to express design knowledge, a single decision may be affected by numerous

constraints. For example in Section 3.5.3, the design type is influenced by a multitude of constraints. Finding out which constraints influence a particular aspect of the design and determining their joint outcome is not a straightforward task and we find this often confuses the domain experts.

Another downside of using IDP is tied to the particular technology used in this system. IDP's model expansion algorithm follows a ground-and-solve strategy (similar to ASP solvers), in which all variables are first replaced by all of their possible values. However, this requires that each variable must have a finite domain, such that the grounding phase can enumerate all of its possible values. Moreover, in order for the grounding to be computed in reasonable time, these domains should be relatively small. Because our application requires some calculations with floating point numbers (e.g., when calculating the shrinkage in cold circumstances), we had to implement a work-around for the normal ground-and-solve workflow.

3.6 Interactive Decision Support

Our goal is to provide design engineers with a decision support system that they can use to interactively explore the possible designs. A graphical interface that allows them to interact with the knowledge base is an essential part of such system.

We reuse and extend the IDP AutoConfig tool [20], which aims to provide a generic knowledge-based configuration tool that can easily be applied in any kind of domain. It supports the principles of the knowledge base paradigm by allowing multiple inference tasks to be executed to the same knowledge. This enables the users to not only perform basic tasks, but also to assist them with a multitude of more advanced questions. The tool displays a three-valued structure which represents the current state of the configuration process. In such a three-valued structure, each ground atomic fact $P(d_1, \ldots, d_n)$ (with P as an n-ary predicate symbol and the d_i domain elements of the appropriate type) or $F(d_1, \ldots, d_n) = d_{n+1}$ (with F an n-ary function symbol) can be either $true$, $false$ or $(unknown)$. In the initial state of the system, all atoms that describe that design are unknown. Once the system is used, increasingly more atoms are assigned a $true$ or $false$ value, either by the user or by the system. This narrows down the final design.

This tool does not force the engineers to use the system in an imposed order (i.e., from input to output). Instead it allows them to specify any piece of information at any stage in the configuration process. When such a (partial) selection is made, this tool computes all its consequences. This way of working is in sharp contrast with how the DMN system can be used.

Because IDP lacks support for non-linear calculations with real numbers, the calculation

of the shrinkage is currently omitted from the knowledge model used in this tool. Moreover, even calculations which are supported may significantly slow down the system. If we include our optimization criterion (see Section 3.5.3) in the theory, the required calculation time for a single propagation step grows to over 20 seconds. For this reason, our current prototype also excludes this criterion. Both limitations originate from the implementation technology used in IDP3 and are not inherent to our approach. We assume that further development to the IDP solver will solve these issues.

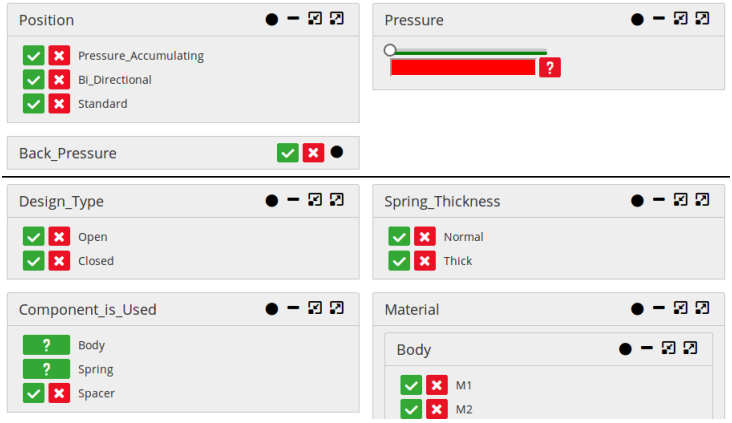
3.6.1 Existing Functionalities

In this section we discuss how we applied the functionalities that were already available in the AutoConfig tool to the context of machine component design.

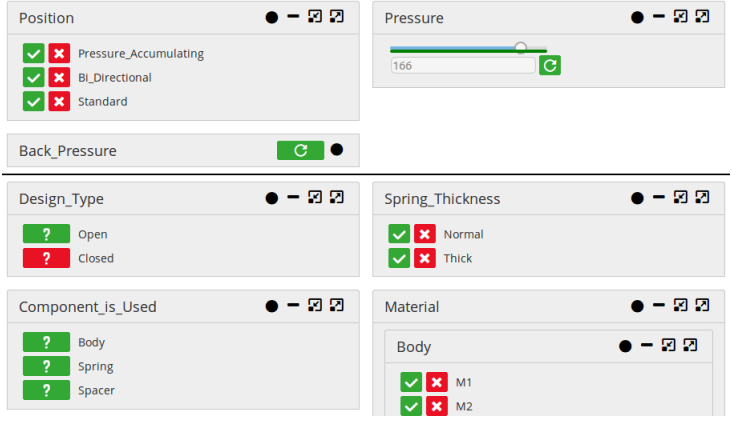
Propagation Once the user assigns a truth value to a certain atom, the consequences of that assignment *propagated* to other atoms. For this, the system considers all two-valued interpretations M that extend the current three-valued interpretation and that are models of the knowledge base T ($M \models T$). If it is the case that a ground atom A holds in all such M, then the truth value of A is switched from $unknown$ to $true$. If A is false in all M, the truth value of A is switched from $unknown$ to $false$.

This functionality is perceived to be very useful in this specific use case, because it allows the system to guide the user in exploring the search space. The system makes a distinction between the user's selection and propagated values: they are indicated with Undo and Question-mark symbols, respectively, as can be seen in Figure 3.8. The reason for this distinction is that they can ask for an explanation of the propagated values.

Explanation For each of the propagated atoms it is possible to request an explanation of why this atom has been propagated as certainly true or certainly false. This explanation consists of the user-selected atoms that have led the system to deduce this truth value. Typically, the user inputs environmental variables under which a design should be able to operate. As a result, some designs can no longer be used.With the explanation functionality, IDP is able to identify which of the input variables, or combination of input variables, leads to that design being infeasible. An example of this can be seen in Figure 3.9. This inference task can be used to explain a certain outcome of the system. This functionality is particularly important for the user acceptance of the proposed product design.



(a) The initial state of the tool.



(b) The state of the tool after selecting Back Pressure and amount of Pressure.

Figure 3.8: Propagation in the AutoConfig tool.

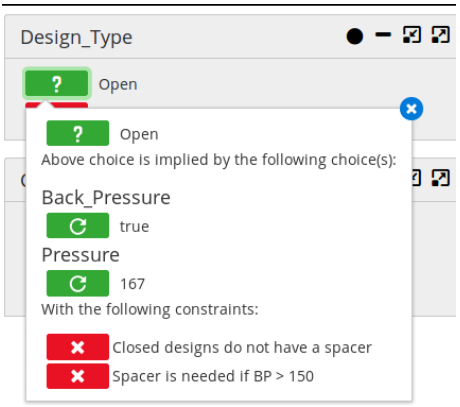


Figure 3.9: Explanation in the AutoConfig tool, consisting of the user-selected atoms (top) and constraints (bottom).

Model expansion Once the user has selected a truth value for a number of atoms, the *model expansion* functionality allows the user to ask the system to complete the current three-valued interpretation into a two-valued model of the knowledge base, by assigning truth values $true$ or $false$ to the $unknown$ atoms in an appropriate way. This can in principle produce any such model and the user has no control over which one is chosen (see Section 3.5.5). In the specific context of machine component design, this functionality did not provide any added value to the AutoConfig tool.

Optimization The AutoConfig tool offers the possibility to optimize a predefined term in the knowledge base, as described in Section 3.5.5. This allows the user to ask the system to compute an optimal model. This feature is useful in the context of product design to determine an optimal design for a specific set of requirements. By introducing multiple terms, it is possible to optimize according to different criteria, such as the lowest production cost, best durability or best overall design.

Relevance The AutoConfig tool offers the ability to specify a *core* of the knowledge base. This core is a set of atoms such that the goal of the configuration process is to determine the truth value of these atoms. Once this has been done, the truth value of the remaining atoms is irrelevant. The relevance functionality allows the AutoConfig tool to determine which predicates and functions may still influence the value of the problem's core, and grey out the irrelevant symbols. This functionality was found useful because it enables them to identify choices that are still relevant to find a solution.

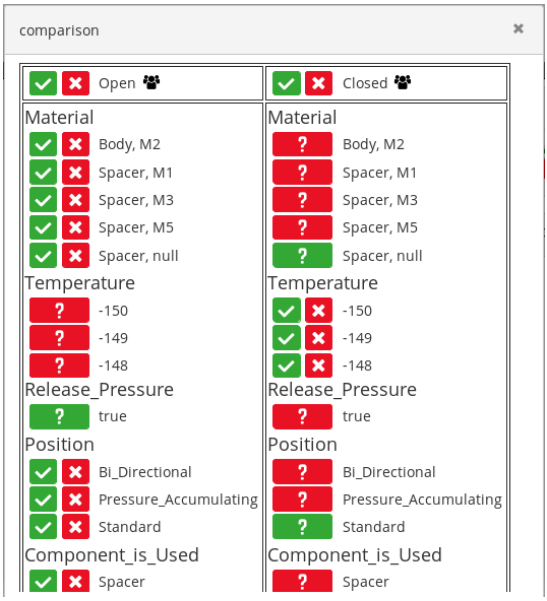


Figure 3.10: Comparison between an open and a closed design in the AutoConfig tool.

3.6.2 Additional Functionalities

Even though the AutoConfig interface was not specifically designed for our product design use case, it proved very useful. However, after testing the tool with the design engineers, we found it was missing some functionalities for this particular case.

Comparison A first functionality the engineers missed was the ability to *compare* multiple designs. Because of the large number of possible designs and materials, it is difficult to know the properties that are associated with each design, especially for less experienced engineers. We developed a comparison functionality, which offers an effective way to identify the key features of a certain design or material. Moreover, when an engineer hesitates between two options for a certain choice, it offers a quick way to see the implications of both options.

Figure 3.10, illustrates our *comparison* method. It shows a comparison made between an open and a closed design. Our method works by first using *propagation* to derive the consequences of each of the possible choices. Then the resulting three-valued structures are compared and all differences are displayed to the user. In the example, the truth value of $ReleasePressure$ is $true$ in the three-valued structure of an open design, while it is $false$ in the three-valued structure of a closed design. In other words, pressure

cannot be released in a closed design, while in an open design, it is always possible to release pressure. Similar to propagation, $true$ and $false$ values are highlighted respectively as green and red boxes. When no truth value can be propagated, it is shown as an unknown value, e.g., in an open design, the *Position* is still open for choice, while in a closed design, the *Bi-Directional* and *Pressure Accumulating* position are not possible, a standard design is the only option.

Extended Explanation The AutoConfig tool already offered the ability to explain why a certain atom is deduced. Originally, this explanation consisted of the set of user-selected atoms from the atom in question can be propagated. However, for the engineers, it was not always clear why these particular selections should have had an effect on that particular atom. Therefore, we extended this functionality by a method to identify the constraints that were fired to make the deduction. This additional information is allows the engineers to better comprehend the reasoning of the system.

As with the atom-based explanation, we invert the truth value of the to-be-explained atom to create an unsatisfiable problem. Subsequently we use the *explain unsat core* inference task of IDP, which is able to determine the smallest subset of constraints that make a problem unsatisfiable.

An additional difficulty is that most engineers lack the ability to read constraints as they are formulated in the knowledge base. To tackle this problem, we allow the knowledge base maintainer to provide a readable annotation for each constraint. A complete explanation thus consists of a set of user-selected variables on the one hand, and a set of annotations of constraints on the other hand. An example of this can be found in the lower part of the explanation in Figure 3.9.

Deactivate constraints It is possible that an experienced engineer has a more thorough knowledge of the domain at hand than the knowledge base. This might be revealed when the system draws an incorrect conclusion. If this occurs, we provide the possibility to temporarily *deactivate a constraint*. Because of the modular nature of the constraints in the knowledge base, the IDP system can still draw conclusions from the remaining knowledge. This functionality enables engineers to experiment with non-standard designs in challenging circumstances.

It only makes sense to start deactivating constraints once the system has derived some atomic property P and the user wants to allow for the possibility that P would not hold. Therefore this functionality can be accessed from the explanation widget, as can be seen in Figure 3.9. Once a constraint is in deactivated state, it is important to visually indicate that it is no longer considered, as can be seen in Figure 3.11, so the engineer does not forget about it and possibly creates an erroneous design.

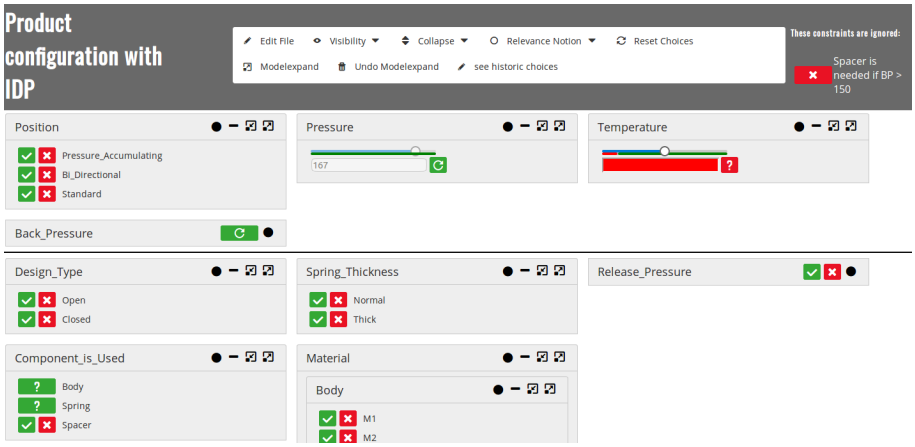


Figure 3.11: Visual representation of when a constraint is deactivated in the AutoConfig tool.

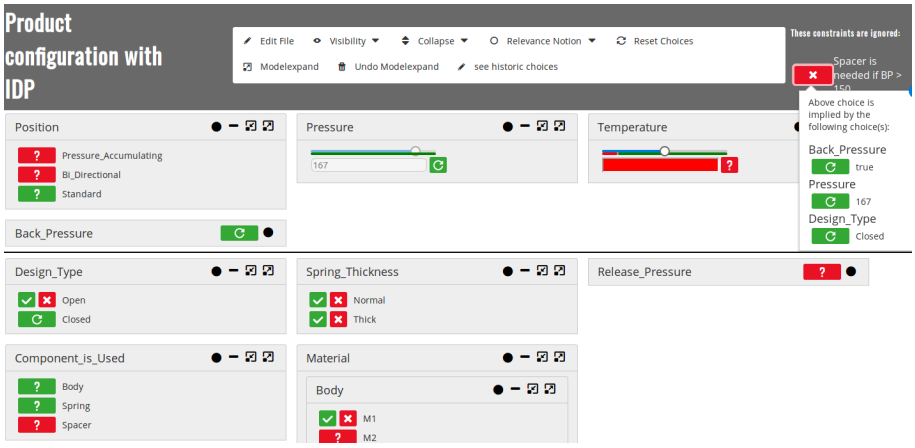


Figure 3.12: Indication of a constraint that cannot be reactivated in the AutoConfig tool

Reactivate constraints After having deactivated a constraint, the user might eventually want to reactivate it, e.g., because he has finished his experimental try-out design. Reactivating a deactivated constraints might create an inconsistent state of the knowledge base. To avoid this, an additional check is performed to ensure a consistent knowledge base. If this check fails, the system alerts the user that it is not possible to perform the action. Similar to the explanation of atoms, an explanation of why the constraint cannot be reactivated is generated, as shown in Figure 3.12.

Provide Feedback When an experienced engineer is using the tool, he might stumble upon a piece of knowledge that is no longer up to date or that should be reviewed, e.g., a constraint that no longer is applicable, or a certain value that should not be possible given the current user-selection. In such situations, the user can report that constraint or atom in order to be reviewed. This report consists of (1) the to be reported element, (2) the current state of the user interface, which enables the problem to be reproduced, and (3) a textual description of the problem at hand, this should be provided by the user. A knowledge base maintainer has to be appointed, who reviews the reports and updates the model based on this feedback. This feedback mechanism ensures that the knowledge base will be kept up-to-date, provided that experienced engineers consistently provide feedback when they encounter such imperfections.

To ensure the correctness of the knowledge base, it is important that updating this knowledge is manually supervised. However, the knowledge base maintainer could be assisted in this task with a number of automated functionalities. It would be interesting to investigate this possibility in future work.

3.6.3 Using Historical Data

The contents of this section are based on joint work with Dries Van Daele and Wannes Meert.

Throughout the years, the company gathered a database which contains all the products that have been designed. This database consists of the customer requirements, technical drawings and design features for each of these products. This historical database contains a lot of knowledge about product design. Presenting the contents of the database in an intuitive way can give the engineers insight in the choices that historically have been made. Next to the constraint-based presentation of the product design knowledge, this historical information can be useful as an additional source of knowledge to support the engineers in their design tasks.

For this purpose, we collaborate with Van Daele and Meert [62]. They developed an algorithm that is able to analyse the design features in the technical drawings and propose a methodology to find similar designs. This methodology defines a similarity measure that is based on the design features that can be found in technical drawings, and some additional database information, such as the customer requirements and a number of supplementary design features.

We use this similarity measure to find designs that resemble the current state of the knowledge base, and present the most similar ones to the user.

In practice, we set up a communication server that provides the necessary functionality for our application. We will refer to this as the ML server. We discuss in detail

the communication with this ML server. Typically, the users first make a number of selections in the AutoConfig tool. The AutoConfig tool then propagates the consequences of these selections, according to the knowledge base. When the user decides to consult the historical database, a web socket connection is initialized with the ML server. After this connection is set up, a request is sent to the ML server. Such a request consists of the following elements.

- *The user's selection*. This is a dictionary that contains a partial structure that only includes the values that the user manually selected. Each of the values can be either certainly true, i.e., the user selected this value to be true, or certainly false, i.e., the user selected this value to be false.
- *The consequences of the user's selection*. This is a dictionary that contains the complete three-valued structure that represents the current state of the user interface. This includes the consequences that the IDP system can draw based on the user's selection.
- *A number of inliers*. This is a list of designs that the user selected as being interesting for his current design task.
- *A number of outliers*. This is a list of designs that the user flagged to be not interesting for his current design task. These inliers and outliers are selected during the Active Learning step (see below).

Based on such request, the ML server sends a response that contains:

- *A list of the three most relevant designs*, each of these designs consists of the technical drawing and its design features.
- *A summary of all relevant designs*. It indicates for each design feature how often this is used in the relevant designs.

Identify Relevant Products

We first discuss how the user's *selection* and *consequences* can be used to make a distinction between relevant and irrelevant designs.

The user's *selection*, *consequences* and each of the designs are mapped on a n-dimensional vector representation, where n is the number of design features. For each of the design features, the corresponding value in this vector can be either true, false or unknown. The similarity between a query vector Q and a design vector D can be measured by the distance between them. This distance is defined as the number of features i that are known in the query vector and have a different value in the evaluated design vector, i.e.,

- When $Q_i = true$ and $D_i \in \{false, unknown\}$, the distance is increased by 1.
- When $Q_i = false$ and $D_i \in \{true, unknown\}$, the distance is increased by 1.
- When $Q_i = unknown$, no distance is added.

A design satisfies a query vector when the distance between them is zero.

There are three possible scenarios. First, the most straightforward case, is when a number of designs satisfy the consequences vector. Because these consequences describe the complete state of the user interface, this means that the ML tool finds a number of historical designs that exactly match the specified selection and, moreover, that also satisfy all constraints in the knowledge base. In this scenario, the relevant designs are those that comply with these *consequences*. The list of three relevant designs is set to any of those matching designs, i.e., each of the relevant designs matches the query equally well.

Second, when none of the designs satisfies the *consequences*, the ML tool evaluates whether there are designs that satisfy the *selection*. In this case, all designs that satisfy the *selection* are considered as relevant. Note that the proposed designs do not satisfy the consequences and therefore contradict the knowledge in the knowledge base. In this scenario there is a distinction that can be made based on the similarity to the consequences vector. The three most similar designs to the consequences vector will be sent in the response.

Third, when none of the designs satisfy the *selection*, all designs are considered relevant, and the three most similar designs are selected based on the distance to the consequences vector.

Currently, when the number of relevant designs determined in these previous steps is too small (i.e., < 3), the list of the three most relevant designs contains duplicates. In future work, a more elegant solution should be determined.

Active Learning

Initially, the three most similar designs are presented to the user, additionally, the summary of all relevant designs is shown. These designs can be labelled as relevant or irrelevant for the current design task.

From this point on, the ML tool provides an active learning mechanism that enables the user to interactively scroll through the relevant designs in a user-friendly way. This method employs a constrained clustering approach, where design similarity is determined by the similarity of their feature vectors. User-provided labels are used to include soft cannot-link constraints between positive and negative examples. Finally,

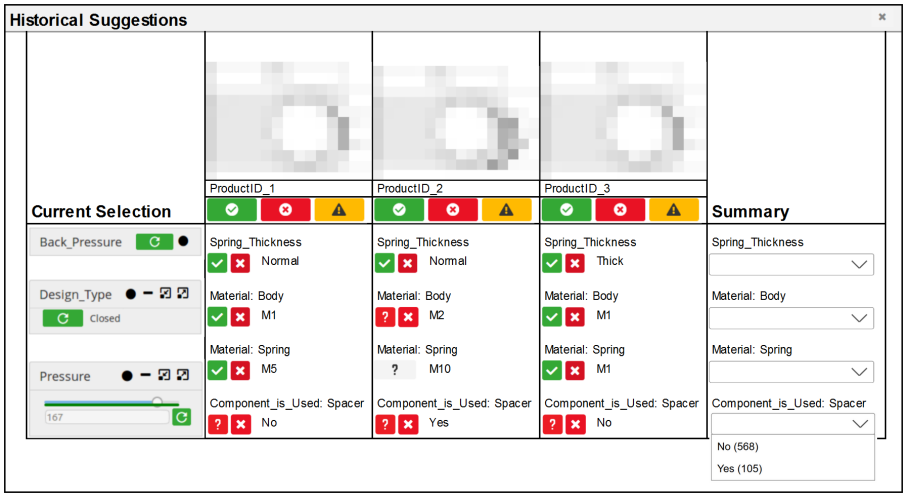


Figure 3.13: Browsing historical products in the AutoConfig tool.

the resulting clustering is used to score each of the designs according to proximity to its cluster centroid, distance from other clusters, and cluster size. The ML server then returns a response containing the three designs collectively considered to be the most informative:

1. The most similar design, based on the positively labelled designs.
2. The medoid of the second most similar cluster (i.e., the most similar cluster that does not contain the first design)
3. The design that would provide the most information when labelled, i.e., a design on the boundary between a positively and negatively labelled cluster.

This step is then repeated, i.e., the user can indicate which designs are relevant or not. This approach allows the user to iteratively explore the historical designs in an intuitive way, and enables him to converge to a certain desired solution.

Presentation

Figure 3.13 shows how these historical products are displayed to the user. We discuss each of its components individually. The *Current Selection* is shown in the leftmost column, it displays the current state of the user interface, i.e., all selections made by the user.

The three middle columns show the *three most relevant products* which are selected as described above. For each of these products, the following aspects are displayed (from top to bottom):

- The technical drawing of that product (we heavily distorted these drawings to make them unrecognizable).
- The product's unique identifier, in case the engineers want to search additional information of this product.
- The buttons for active learning. These buttons can be used to either label the product as interesting, label the product as irrelevant or report this product as outdated. This last option enables to keep the product database up-to-date. When one of these buttons is clicked, a new request is sent to the ML server, with updated inliers and outliers.
- The design properties of that product. There are four options: (1) the underlying knowledge model has made no deductions about this property, i.e., it can be either true or false in the current state of the knowledge base. In this case, it is possible to select or deselect this property. E.g., $BodyMaterial = M_1$ could be either true or false in the current state of the knowledge base. (2) The underlying knowledge model has already deduced this property. In this case, the possible actions are to either select this property or ask an explanation of why the model has deduced this property. (3) The property contradicts the knowledge in the underlying model, e.g., the second design has M_2 as $BodyMaterial$. In this case, the possible actions are to deselect this property, or to ask an explanation of why the model has excluded this property. (4) Lastly, when the underlying knowledge model has no knowledge at all about a certain property, e.g., material M_{10} is not known in the knowledge base. In this scenario, the grey rectangle indicates that no actions can be performed.

  When the user makes an additional selection, the *propagation* functionality is called, and a new request is sent to the ML server, with updated *selection* and *consequences*.

The rightmost column in Figure 3.13 displays a *summary* of all relevant designs, which are selected as described in the section on how to Identify Relevant Products. For each design property, it is indicated how frequently it appeared in the relevant products.

3.6.4 Evaluation

Evaluation Procedure

During the research project, some engineers from the case study company were solicited multiple times to detect areas for improvement. Over the course of several months, they got familiar with bits and pieces of the interface and its functionalities. After incorporation of the suggested functionalities in the AutoConfig tool, we organized an evaluation workshop to find out if the project goals were met. Five engineers with different levels of exposure to the tool and of experience in the tested application area were invited to test the AutoConfig tool. Their experience ranged from a couple of months to numerous years. Some of the subjected engineers had been closely involved in the project, while for others it was the first time they used the tool. Due to the COVID-19 measures, the workshop was organized online.

The company's project manager introduced the workshop to the attendees, recalling the purpose of the tool and the session. Then a researcher demonstrated and explained the different functionalities of the tool. Subsequently the engineers had 15 minutes to play around with the tool themselves and ask questions. Then, they received an assignment, i.e., a set of requirements for which they had to design a component using our tool, and an evaluation form they would need to fill in afterwards. After the assignment, each engineer individually filled in the questionnaire and sent it to the researchers. The results of the different engineers were compared and taken as a starting point for a group discussion. The evaluation consists of three large parts: we consider the expected usage of the tool, its benefits, and its potential target users. The results of these three parts are described below.

Usage of the AutoConfig Tool

The tool was developed to support engineers with the design of non-standard products. In our evaluation, we wanted to check whether the engineers indeed found it useful for this purpose. In their answers, it turned out that the engineers would indeed use the tool to this end. They thought the tool would be most useful for designs that are not too standard, but not too specialized either. On the one hand, for very common designs, experienced engineers might immediately know the right design and perceive the tool as overhead. However, they pointed out that, even in such a case, the tool might still be useful as a way of possibly challenging their assumptions and ensuring standardization. On the other hand, for designs with complex geometry, even after using the tool, the engineer would still need to make manual calculations, such as fine-tuning the dimensions of the components. In the follow-up discussion, the engineers indicated that they saw the manual calculations as inevitable, and not a disadvantage of the tool. The search for historical designs was seen as an interesting functionality

of the tool. Finally, the engineers also thought the tool would be useful for training purposes and double checking the personal expertise.

Analysis of product failures was also mentioned as a potential application for the tool. It could be useful to enter the failed product and the operating conditions at the time of failure into the AutoConfig tool, and check which constraints are violated. However, this is a cumbersome way to analyse failures and could be improved by adding a specific functionality for this.

For the tool to remain useful, more experienced engineers need to use it regularly, and update it when they notice unsound or outdated constraints. It is therefore important that the company incentivise its use and stress its importance in promoting consistent designs and validating one's assumptions.

Benefits of the AutoConfig Tool

The second part of the questionnaire focusses on the expected benefits of the system. When asked about the daily time benefit, the engineers report an expected daily gain of at least 15 minutes to more than 30 minutes per engineer, depending on the number of incoming requests per day. Moreover, they expect to replace the daily design meeting by this tool. The estimated time benefit takes into account the time it takes to get used to the tool, the effort to update the tool and the fact that experienced engineers are able to develop more regular products without the tool. The tool takes a few seconds to show the consequences of a choice, or to retrieve designs from the databases. One of the engineers commented that the tool would be more usable if it was more responsive.

For some tasks the tool is useful and convenient, but does not translate into a quantifiable time gain. For example, even though the tool helps to consult historical designs, this remains a time consuming task. The reason is that the historical database contains a large number of products. While the used ML techniques attempt to select the most relevant ones, this may still be a large list.

Another important benefit is the expected increase in "first time right" designs : this lowers the production time and cost, and increases customer confidence. It also makes sure designs are more consistent, which in turn increases customer confidence. Nowadays, under similar circumstances, different designs might be proposed by the US plant and by the European plant. Unsurprisingly, customers find this very strange.

Other benefits mentioned include the continuous learning about designs and their constraints, and the promotion of a standard to judge design quality.

Target Audience of the AutoConfig Tool

The usefulness of the tool might be different for engineers with different levels of experience. From the questionnaires, it turned out that the test persons think the tool will be equally useful for engineers with more experience, and even more useful for less experienced engineers. In the discussion that followed, it appeared that the tool will probably be used differently depending on the level of experience. It can be seen as a knowledge transfer tool from more experienced to less experienced engineers. For the first group, the tool is not only useful to help with the design process, but especially to challenge one's own assumptions and design preferences. This group is expected to contribute heavily to the development and maintenance of the tool. For the latter group, the tool will be an indispensable support for their day-to-day design tasks, and will be part of their training.

Conclusion

Although the engineers were consulted during the development process of the AutoConfig tool, we were surprised by the positive reactions during the evaluation session. For the design process the main advantages that surfaced are the time gain of an estimated 10%, improved consistency and accuracy in designs. The combination between a knowledge based approach that lines out the constraints of the design, and the possibility to review historical designs, is much appreciated. Additionally, the tool is not only relevant for the design of a specific product, it can more widely be seen as a knowledge transfer and training tool. For novice engineers, a wealth of knowledge is available. For experienced engineers, the interaction with the tool challenges the assumptions and practices of the engineer, while the engineer challenges and updates the constraints in the knowledge base. This motivates experienced engineers to use the tool, even if they could design regular products without it. From the overall feedback, we conclude that the AutoConfig tool in its current format meets the requirements that were identified at the beginning of the project.

3.7 cDMN: A user-friendly representation of constraint knowledge

The FO($\cdot$) representation with constraints has a clear advantage over the DMN decision process model in terms of versatility and applicability. The expressive language allows to represent complex knowledge, and the constraint-based reasoning and related inference tasks allow the applicability in multiple use cases. The largest challenge for the FO-based constraint approach, is the extraction of the actual domain knowledge

Types

Name	Type	Values
Component	string	Body, Spring, Spacer
Material	string	M_1, M_2, M_3, M_5, M_4
temperature	int	[-200,200]
Dimension	float	[0, 10]
...	...	...

Constant

Name	Type
CS	Dimension
ID	Dimension
OD	Dimension
Temperature	temperature
...	...

Boolean

Name
Release Pressure
Back Pressure

Function

Name	Type
Material of Component	Material
MinTemp of Material	temperature
MaxTemp of Material	temperature

Relation

Name
Component is Used

Figure 3.14: Extract of the cDMN glossary for the product design application.

from domain experts. Not only is it often hard to formulate tacit knowledge, we currently also lack a formal language that allows domain experts to understand the knowledge captured in the constraint model. This opens the way for misunderstandings, and prevents the domain experts from working on the model without the presence of a knowledge engineer. In well-delineated cases the DMN standard proved to be very useful to tackle this challenge. This stands in stark contrast to our knowledge elicitation methodology for the decision process for the well-known applications, in which the DMN standard helped us to avoid these problems.

In this section, we investigate how the cDMN notation introduced in Chapter 2 can be used to represent the knowledge in the running example introduced in Section 3.3. The cDMN notation represents constraints in a table format. Consequently, the knowledge in the cDMN model will be almost identical to that of the constraint model discussed in Section 3.5.3.

3.7.1 Glossary

An extract of the cDMN glossary of our running example can be found in Figure 3.14. It describes the vocabulary that can be used throughout the cDMN model.

3.7.2 Dimensions

Because we transform cDMN tables into FO(·) formulas that we give to IDP, our system can propagate the value of any arbitrary subset of variables in such a table to the other variables. In other words, the distinction between “input” and “output” variables may still be a useful way for domain experts to think about the structure of a table, but no longer has implications for how the table can be used by the system. Consequently,

Dimensions	
U	**CS**
1	(OD- ID)/2

Figure 3.15: Decision table for dimensions.

Release pressure			
E*	**Design Type**	**Position**	**Release Pressure**
1	Open	–	Yes
2	Closed	–	No
3	–	Pressure Accumulating	Yes

(a) Constraints related to Release Pressure.

Spacer is Used					
E*	**Design Type**	**Position**	**Back Pressure**	**Pressure**	**Spacer is Used**
1	Closed	–	–	–	No
2	–	Bi Directional	–	–	Yes
3	–	–	Yes	> 150	Yes

(b) Constraints related to the use of a spacer.

Use Component		
E*	**Component**	**Component is Used**
1	Body, Spring	Yes

(c) Constraint table to specify Body and Spring are always used.

Figure 3.16: Constraint tables for the selection of Design Type and Use Component.

the relation between component dimensions can easily be represented in a decision table with a single row (Figure 3.15).

3.7.3 Design Type and Spacer

The constraints considering *design type* and *spacer* described in Section 3.5.3 can be easily put into two constraint tables with independent concerns, one for the concept of *Release Pressure* in Figure 3.16a and one for the use of a *Spacer* in Figure 3.16b. The formulation of knowledge in constraint tables leaves considerable freedom in selecting a design and spacer. Section 3.7.6 discusses how this is handled.

Note that the constraint table in Figure 3.16a contains both descriptive (e.g., open designs can release pressure) and prescriptive information (e.g., in pressure accumulating positions, pressure should be released), as discussed in Section 3.5.1.

Component Materials				
E*	**Component**	**Component is Used**	**Design Type**	**Material of Component**
1	Body	Yes	–	M_1, M_2, M_3
2	Spring , Spacer	Yes	–	M_1, M_3, M_5
3	–	No	–	null
4	Body	Yes	Closed	Not(M_2)

(a) General constraints on components' materials.

Temperature Constraints			
E*	**Component**	**Component is used**	**Temperature**
1	–	Yes	$\geq$ MinTemp of Material of Component
2	–	Yes	$\leq$ MaxTemp of Material of Component

(b) Temperature restrictions on the selection of materials.

Data Table: Temperatures			
	Material	**MinTemp of Material**	**MaxTemp of Material**
1	M_1	-150	100
2	M_2	-120	80
3	M_3	-50	200
4	M_5	-50	150

(c) A data table to define MinTemp and MaxTemp of each material.

Figure 3.17: Tables for the selection of component materials.

3.7.4 Materials

Figure 3.17a specifies for each of the components which materials can be used, e.g., the body can only be constructed from material M_1, M_2 or M_3. Of course, this constraint is only valid if the component is used in the product. When this is not the case, i.e., a component is not used, its material is set to null. The last line in Figure 3.17a states the additional constraint that the body of a closed design cannot be made of material M_2.

In addition to these general constraints on the materials of all components, the materials each have a minimum/maximum temperature. Those temperatures are listed in a data table (Figure 3.17c). The table in Figure 3.17b states that, for each component that is actually used in the design, a material must be selected such that the operating temperature falls within the temperature range of this material.

As we discussed in Section 3.5.3, these constraints still leave considerable freedom in selecting a material. We will discuss how to prioritize certain materials in Section 3.7.6.

3.7.5 Shrinkage check

The decision tables in Figure 3.18 describe how to determine the *Shrinkage Load* and *Spring Strength* and state the constraint that the former should not exceed the latter.

Shrinkage Load		
U	**Design Type**	**Shrinkage Load**
1	Open	f(CS, Material of Body, Temperature)
1	Closed	f(CS, Material of Body, Temperature)

(a) Calculation of Shrinkage Load.

Spring Strength				
U	**CS**	**Design Type**	**Spring Thickness**	**Spring Strength**
1	[0.75, 1.25]	Closed	Standard	15
2		Closed	Thick	20
3		Open	Standard	7
4		Open	Thick	11
5	[1.75, 2.25]	Closed	Standard	13
6		Closed	Thick	18
7		Open	Standard	6
8		Open	Thick	9

(b) Decision table to define Spring Strength.

Shrinkage Check	
E*	**Shrinkage Load**
1	$\leq$ Spring Strength

(c) Constraint table for the shrinkage check.

Figure 3.18: Constraint approach to determine whether the shrinkage load can be handled.

Note that this is a more intuitive way of representing the knowledge in Figure 3.7.

It enables us to differentiate between the constraint (i.e., *Shrinkage Load* should not exceed *Spring Strength*), and how to determine these values based on other design properties. As discussed in Section 3.4.2, a workaround is needed in DMN to obtain a similar result, i.e., although the *Spring Strength* is a property of a spring with a certain thickness, the *Spring Thickness* needed to be represented as an output of the DMN table that checks shrinkage, for lack of better solutions. As mentioned in Section 3.5.7, the *Shrinkage Load* and *Spring Strength* are also influenced by the *Design Type*. Because cDMN's constraint-based nature, a switch from an open to a closed design is considered if the *Shrinkage Check* cannot be satisfied with an open design, we were not able to achieve this behaviour in standard DMN.

3.7.6 Preferences

Because of the constraint-based nature of cDMN, typically, a cDMN model will not have a unique solution. If we provide the system with a set of dimensions and operating conditions, there may be many possible combinations of designs and materials that meet all of the constraints. Usually, the user is interested in finding an optimum in this

set of possible solutions. For instance, the product engineers try to find the cheapest design or the most reliable design. Such global optimization is not possible in regular DMN.

Figure 3.19 shows how an optimal design can be selected, based on a number of parameters. The underlying reasons behind these tables are identical to those defined in the *preferences* in Section 3.5.3.

Design Selection

We select a number of examples to highlight the alignment between the DMN tables in Section 3.4.2 and the cDMN model.

When both open and closed designs can be used, the leak tightness of closed designs is superior (Figure 3.19d). The cDMN model therefore selects a closed design when maximizing the optimization term, this is indeed what also happens in the DMN table in Figure 3.5a.

Thick springs have a higher cost than standard springs, as can be seen in Figure 3.19b, so they will only be selected when a high shrinkage force demands to use such a thick spring.

Figure 3.19c states that the durability of a design is gravely affected when no spacer is used in a high back pressure (≥ 100) environment. In such circumstances, this design will never be used if a spacer design is available. This behaviour can also be found in row 4 and 5 of Figure 3.5a, where a closed design without spacer is only used in low temperature (i.e., high shrinkage) circumstances.

In contrast, when no spacer is used in lower back pressure environments, the cost difference (i.e., designs without spacer are cheaper) outweighs the loss in durability. When the weights of the optimization function are adjusted, this could shift the preferences in this case.

Materials

Because of the penalty for using locally unavailable materials (Figure 3.19e), the cDMN model never selects material M_4. When this material would be available again, the adaptation to the cDMN model is trivial (i.e., changing the availability in the Material Properties data table), while the DMN model should be completely adapted.

When the temperature is between -50 and 100 degrees, each material could be used for the spring. When taking in account the cost of each of the materials, specified in

Data Table: Material Properties			
	Material	**Cost of Material**	**Material is Locally Available**
1	M_1	8	Yes
2	M_2	5	Yes
3	M_3	10	Yes
4	M_5	4	Yes
5	M_4	7	No

(a) Data table: Define cost and local availability of materials.

Cost				
C+	**Component called c**	**c is Used**	**Spring Thickness**	**Cost**
1	–	Yes	–	Cost of Material of c
2	–	–	Thick	5
3	–	–	Standard	2

(b) Define total cost of design.

Durability				
C+	**Spacer is used**	**Back Pressure**	**Pressure**	**Durability**
1	No	Yes	< 100	-2
2	No	Yes	≥ 100	-100
3	–	–	–	$-1/5\times$ Spring Strength

(c) Define durability of design.

Leak Tightness	
U	**Leak Tightness**
1	$1/2\times$Spring Strength

(d) Define leak tightness of design.

Availability				
C+	**Component**	**Component is used**	**Material of Component is Locally Available**	**Availability**
1	–	Yes	No	-100

(e) Define availability penalty.

Execute
Maximize Durability + Leak Tightness + Availability - $5\times$ Cost

(f) The optimization term that will be optimized when executing the model

Figure 3.19: Preferences in cDMN.

Figure 3.19a, cDMN selects the cheapest possible material, i.e., M_5. This is indeed also the material selected by DMN in Figure 3.6b.

3.7.7 Discussion

We compare the cDMN model of our running example with the FO(·) representation presented in Section 3.5.3. A first conclusion we can draw is that the proposed notation is expressive enough to represent the knowledge in this running example, i.e., all knowledge fits well within this table-based notation. Second, the readability of the knowledge model considerably improves by expressing the knowledge in the cDMN language. Not only does the cDMN notation allow the decision logic to be expressed in a readable table-based notation, it also allows data to be expressed in a straightforward user-friendly manner (i.e., typically data is already available in similar tables).

We also make a comparison of the DMN model discussed in Section 3.4.2 and the cDMN model in this section. A first thing one might notice is that the cDMN model is considerably larger than the standard DMN model, i.e., the DMN model consists of only nine decision tables, whereas the cDMN model consists of ten tables that determine the physical constraints and an additional six tables to express the possible preferences. The larger size of the cDMN model is compensated for in terms of practicality:

- No decision procedure should be determined prior to creating a cDMN model, this is particularly useful when such procedure is not at hand.
- The cDMN model embodies inherently more knowledge than the DMN model, i.e., the DMN model contains only knowledge about how to determine one output based on a set of inputs, it contains no information of alternative solutions, nor the reason why a solution is selected.
- Consequently, the knowledge in the cDMN model can be used in a lot of additional ways.
- The knowledge expressed in a cDMN model is much more modular. When a certain preference, constraint or value of a property changes, this can easily be adapted without changing the rest of the model.
- As a result of introducing multiple optimization criteria, the same model can be used when, in a certain request, the priorities deviate from the standard priorities. In regular DMN, a new model should be created for each possible optimization function.

Even though the cDMN model is larger, the readability of the model is not affected. Each of the tables is of equal legibility as its DMN counterparts.

3.8 Related Work

Product Configuration There exists a vast body of research on the topic of automatic product configuration, which is typically defined as the task of automatically constructing a design from a set of pre-defined components, considering several constraints and some optimization criteria [13]. Research shows that product configurators have a positive impact on lead time [37, 30] and sales quotation time [41]. Other comparison studies [61] investigate the effect of configuration systems on product quality, also showing promising results.

A thorough literature review on product configuration was performed by [71]. Their findings reveal that, despite the wide range of existing research, several topics still require further exploration. First, although knowledge acquisition from historical data has been extensively studied, less research has been done on extracting knowledge from domain experts. Moreover, knowledge representation research typically focuses on methods that are intended to be used by knowledge engineers. Little attention has been paid to representations that are usable by domain experts. Our work examines the use of DMN and its extensions to cDMN to address these issues in the context of our concrete application domain.

A second aspect which according to Zhang [71] has not yet received much attention is the ability to suggest new designs. The majority of existing product configuration approaches focus on selecting the most appropriate option among a fixed range of possibilities. By contrast, our constraint-based approach is also able to provide useful information to the engineers in cases that fall outside the scope of existing solutions.

Third, Zhang [71] also identifies several ways in which additional forms of inference might be useful to provide functionality other than suggesting a design. For example, she identifies such tasks as explaining which conflicting constraints have led to a rejected design or reconfiguring an existing design to cope with changed requirements. The IDP system has been developed according to the knowledge base paradigm [25], in which different logical inference methods can be applied to the same knowledge base in order to implement different functionalities. Indeed, in Section 3.6, we describe how the AutoConfig tool can be used in an interactive way, which, amongst other functionalities, contains the ability to explain the system's behaviour. Moreover, the task of reconfiguration (i.e., model revision) has already been considered in the context of IDP [42]. The IDP system therefore provides a suitable formalism to express the design knowledge.

DMN The fact that DMN decision tables are often too limited for real-life cases has been recognized before [15, 26]. In addition to the S-FEEL language that we have used in Chapter 2, the DMN standard itself also puts forward the more general FEEL

language to allow more complex expressions. However, FEEL is a programming language with its own syntax, and hence not suitable for direct use by domain experts [52].

A number of DMN tools provide ways to deal with the limitations. For instance, OpenRules allows to insert Java-snippets to express complex parts of the logic. The advantage of this approach is that it still keeps the overall idea of decision tables that can be maintained by business experts, while allowing an IT-expert to code specific complex parts of the decision logic.

In addition to allowing imperative code snippets, several approaches also allow DMN to be extended by more declarative representations. For instance, the aforementioned OpenRules also offers an interface to a constraint solver [29], while [15] allows DMN to be enriched with domain knowledge expressed in Description Logic. The latter paper demonstrates its approach by means of a case study of port clearance for cargo ships. Here, the clearance decision itself can be modelled with traditional DMN, but the ontology of cargo ship types and their features are modelled in the Description Logic.

These papers share with our work on cDMN the goal of extending DMN to be able to cope with the complex knowledge that typically arises in real-world problems. However, they extend DMN's decision tables with a completely separate formalism, that is intended to be used by knowledge engineers rather than DMN approach. By contrast, our goal is to investigate whether it is possible to allow the more complex knowledge to also be expressed within DMN's table format, allowing it to be maintained by domain experts as well.

3.9 Conclusions and future work

In this chapter, we explored multiple possible approaches to develop a decision support system for the design of machine components. This research was conducted in collaboration with a multinational company that aims to standardize and partially automate its design process. The products that need to be designed can either follow a routine design process, if they are to be used in known application areas, or an ad hoc design process, if they will be used in new and challenging application areas.

This project's main challenge is that there are two potentially contradictory requirements. On the one hand, a flexible and powerful knowledge representation method will be needed to allow useful conclusions to be provided to the engineers even in circumstances that fall outside of their designs' usual scope. On the other hand, the company needs a system that can deterministically propose standard designs, to assist the people in the sales department. Moreover, the engineers need to be closely

involved in the formal specification, since they are expected to agree on and understand the model, and to help maintain it.

In this light, we modelled knowledge in both a deterministic and constraint-based way, using the DMN standard for the former and the IDP system for the latter.

The DMN-based approach has proven to be very useful and is currently used by the company in their production environment.

To assist the engineers in more challenging application areas, we allow them to interact with a system that presents a multitude of user-friendly functionalities. We based this system on the IDP AutoConfig tool, which we extended with a number of functionalities, to assist the engineers in even more ways. Next to these knowledge-based functionalities, a number of ways to explore the historical data were added to this tool. The engineers indicate that the proposed system is indeed very useful in a number of ways, by saving them a lot of time, reducing manual errors, helping them converge to a more standardized design proposal and it could even be used as a training tool for less experienced engineers. Moreover, because the system allows the users to provide feedback, which is used to update outdated knowledge in the system, it also increases confidence in the system.

While both the DMN-based and constraint-based approach have their advantages, neither is able to completely fulfil all of our requirements. We therefore propose the cDMN framework to combine the user friendly notation of DMN with the more versatile functionalities that can be provided by the constraint-based approach.

In future work, we will look into a more standardized approach to represent missing data (e.g. when material properties are not known). Moreover, it would be interesting to investigate whether historical data can be used to identify additional constraint to supplement those in the knowledge base. An additional possibility is to extract knowledge from the general constraint-based knowledge base, to automatically create or update the deterministic procedure of a certain sub-application, and to keep knowledge in these two systems up-to-date. Currently, the constructed knowledge base should be maintained manually, it could be useful to find a way to assist the knowledge base maintainer in this process. By extending the knowledge in the constraint-based knowledge base, more general design rules can be determined, which will improve the applicability of the system. Finally, the IDP system is not able to perform floating point calculations, therefore we are looking into the possibility to implement the knowledge base in a system that can handle these calculations.

Chapter 4

A Probabilistic Logic Programming Approach to Automatic Video Montage

In this chapter we propose an automatic video montage system. Creating a pleasing video montage is not a deterministic process, because multiple montages of the same footage could be of equally high quality. We therefore propose to represent this knowledge in Probabilistic Logic Programming (PLP). This offers us the opportunity to explore the probabilistic side of knowledge representation.

Figure 4.1 indicates the research topics on which we focus in this chapter. First, we construct a cinematographic model from expert know-how, which contains a multitude of cinematographic rules. We then investigate how this model can be used to produce video montages in semi real-time. Because of the graphical nature of the problem domain, presenting the results to the user is straightforward, i.e., when the raw video streams are concatenated to a montage, the resulting video can be readily presented to the user of the system. We then show that a readable representation of the cinematographic knowledge can be created using a probabilistic variant of the cDMN notation, presented in Chapter 2. The cinematographic model is based on expert know-how. This model consists of a number of crisp rules, and a number of probabilistic rules. We attempt to use data to fine-tune the constructed model, by calibrating the used probabilistic parameters.

The research in this chapter was funded by the KU Leuven Research Fund as part of the GOA project “CAMETRON”. The content of this chapter is based on publications in the proceedings of the ECAI 2016 conference [4], the WICED 2017 workshop [5]

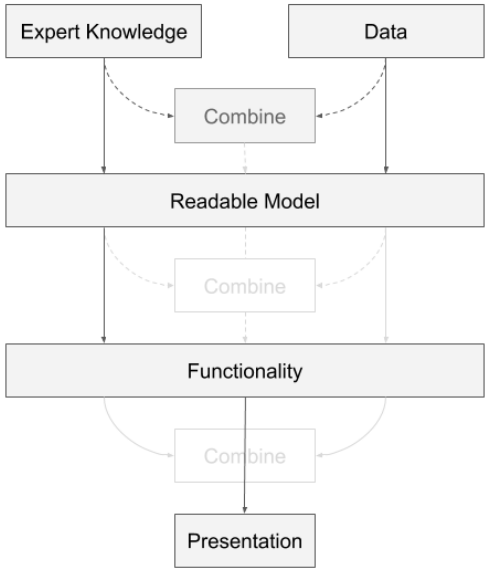


Figure 4.1: Visualization of the research topics in this chapter.

and joint work with Dries Hulens et al. at the MMM 2018 conference [39].

I would like to thank Dries Hulens for providing the video analysis module used in this work.

4.1 Introduction

Events such as lectures, sports games, musical performances, etc. are often recorded on video. When produced by a professional video crew, these recordings are typically of high quality, but also quite expensive. Because good-quality cameras are no longer overly expensive, organizers may be tempted to instead produce a "home-made" recording of their event, filmed by amateurs. However, while this option is much cheaper, the end results tend to be of noticeably lower quality than those produced by a professional crew.

The research described in this chapter fits in the CAMETRON project. The goal of this project was to fill the gap between videos produced by a professional crew and home-made productions, by creating a system to produce recordings whose quality approaches that of a professionally produced video, but without the price tag. It will

rely on an almost completely automatic system, thereby eliminating the expensive personnel cost. Our goal is to produce full-length reports of events, i.e., videos that span the entire length of the event itself, rather that a summary or a set of highlights.

The CAMETRON system consists of two separate components, mimicking the roles found in a typical film crew: virtual "camera men" (consisting of fixed cameras, pan-tilt-zoom cameras and cameras mounted on UAVs, together with the software to operate them) capture the footage, while a virtual "editor" combines footage of the different cameras. The development of the first component has been described elsewhere [40, 23, 39]. In this chapter, we present the second component: a virtual editor system, that is able to produce a single coherent, qualitative video from a number of different feeds of raw footage from the same event.

Creating a video that is both interesting and easy-to-follow is not a straightforward task. Human editors typically follow a number of different—and sometimes contradictory—cinematographic "rules" to accomplish this task. To develop our virtual editor, we will follow a declarative approach, in which we explicitly represent these rules. This approach has the benefit that it offers a great deal of flexibility in deciding which rules should be taken into account and how they should take priority over each other. An additional advantage is that it also allows us to reuse the same knowledge to perform different tasks: we cannot only use the rules to generate a montage, but also to evaluate the quality of a given montage or to learn certain properties of good montages from given examples.

To represent the rules, we need a suitable knowledge representation language. A particular challenge in this application is that cinematographic rules are not strict: they are guidelines that are typically followed, but not always. Indeed, the rules may sometimes contradict each other, and even if they do not, a human editor may still choose to ignore a rule, simply because the result "feels" better. A virtual editor should therefore not rigidly follow the rules, but it should sometimes deviate from them in order to give the montage a more interesting and natural flavour, thereby mimicking the creativity of a human editor. For this reason, we have chosen to make use of a Probabilistic Logic Programming (PLP) language, which allows us to represent these rules in a non-deterministic way. This has the additional benefit that—just like a human editor—the system is able to produce different montages from the same input streams.

We restrict attention to videos of relatively simple settings, like a lecture, a sports event or a concert. More challenging settings, such as fiction, are left for future work. Our work differs from existing approaches in various ways. First, unlike techniques that construct summaries of events [34, 32], our method produces a video of the complete event. Second, unlike approaches that are developed for a specific setting [46, 58, 33], we do not assume a fixed number of cameras, a fixed camera setup or a particular subject matter. Third, our system assumes as input only video footage, in contrast to, e.g., the virtual editor systems that are present in game design and which use a 3D model of

the scene [18, 38]. Fourth, our system is able to make the required editing decisions in real-time (for 25fps video), unlike, e.g., [19, 43]. Fifth, we incorporate a variety of different cinematographic rules into our system and demonstrate that the resulting montages are almost indistinguishable from those produced by a professional editor, by performing a Turing test. Existing work either does not perform such validation [43, 58, 33] or appears to have worse results than our method [19, 46], as we discuss in Section 4.11. Finally, systems such as [19, 43] are deterministic, in the sense that they always produce the same montage for the same video stream. The fact that our approach is more flexible in this respect may prove useful if the user for some reason does not like the montage initially proposed by the system.

The remainder of this chapter is structured as follows. We describe ProbLog, the PLP system that we will use, in Section 4.2. In Section 4.3, we discuss a number of cinematographic rules. Section 4.4 describes our editing system in detail, in Section 4.5 we describe how this system can be used. Section 4.6 discusses the computational performance of this system. In Section 4.7 we discuss the influence of the used probabilities on the quality of the montage. Section 4.8 investigates the quality of produced montages. We use the cDMN framework to represent the non-deterministic knowledge in Section 4.9. Section 4.10 discusses the option to use data to fine-tune the probabilities in the cinematographic model. In Section 4.11, related work is discussed in more detail. We conclude in Section 4.12.

4.2 Preliminaries: CP-logic and the ProbLog System

CP-logic [66] is an expressive PLP language, based on Sato's distribution semantics [59]. A theory in CP-logic consists of a set of rules of the following form:

$$\alpha_0 :: A_0 \quad ; \quad \cdots \quad ; \quad \alpha_n :: A_n \qquad :- \qquad \phi. \tag{4.1}$$

Here, α_i are probabilities with $\sum_i \alpha_i \leq 1$. A_i are atoms, i.e., predicates applied to their arguments, such as $throws(billy)$. Lastly, ϕ is a logical formula, typically a conjunction of atoms (or negations of atoms). Each such rule represents a non-deterministic causal mechanism: when the body ϕ is true, it triggers a non-deterministic event which causes at most one of the A_i; for each i, α_i is the probability that A_i is the outcome of this event.

We first consider only the ground case, in which no variables are allowed. In this case, the formal semantics of CP-logic can be characterized in terms of the well-founded semantics for normal logic programs as follows. Suppose that, for a rule of form (4.1), we probabilistically either replace this rule by one of the rules $A_i :- \phi$, each with probability α_i, or we remove this rule altogether with probability $1 - \sum_i \alpha_i$.

If we do this independently for each of the rules r in a CP-logic theory T, then we probabilistically reduce T to a normal logic program P. By π_T, we denote the resulting probability distribution over these normal logic programs P. The formal semantics of the CP-logic theory is then the probability distribution P_T over possible worlds (i.e., Herbrand interpretations) that is defined by $\pi^*_T(I) = \sum_{P \models I} \pi_T(P)$; here, the sum is taken over all normal logic programs P that have the interpretation I as their well-founded model. As usual, for a formula ϕ, we define the probability $\pi^*_T(\phi)$ as $\sum_{I \models \phi} \pi^*_T(I)$. This semantics is only well-defined for theories T that have the property that all logic programs P that can be produced from them (with non-zero probability) have a two-valued well-founded model (which is then also the unique stable model of P). CP-logic disallows theories that do not have this property. [1]

A small example is the following CP-logic theory T_{SB}. It describes two children, Billy and Suzy, who each may decide to throw a rock at a bottle. Both children throw with 75% accuracy.

```
0.5::throws(billy).
0.6::throws(suzy).
0.75::breaks :- throws(billy).
0.75::breaks :- throws(suzy).
```

According to this theory, for instance $\pi^*_{T_{SB}}(breaks) =$
$0.5 \times 0.6 \times 0.75 + 0.5 \times 0.4 \times 0.75 + 0.5 \times 0.6 \times (1 - (1 - 0.75)^2)$
$= 0.65625$.

These semantics are extended to the non-ground case by viewing a non-ground rule as a template for the set of all of its ground instantiations. This approach is identical to how variables are treated in Answer Set Programming [47, 51]. It requires all rules to be finitely groundable. This can be ensured by disallowing function symbols and requiring that all variables must appear in a positive body atom.

The above example is therefore equivalent to:

```
0.5::throws(billy).
0.6::throws(suzy).
0.75::breaks :- throws(_).
```

The anonymous variable "_" in the last rule indicates that this rule will fire for each person that throws.

The ProbLog system [22] is a state-of-the-art inference system for amongst others CP-logic. Its Prolog-based input language supports standard Prolog, as well as CP-

[1] If any such program had a three valued well-founded model, then $\sum_I \pi^*_T(I)$ would be < 1, i.e., there would be missing probability mass.

logic-like rules. In particular, it allows lists, the use of predicates such as `findall` and the use of variables as probability labels in the head. It supports inference tasks such as querying the probability $\pi_T^*(A)$ of an atom A and sampling a particular interpretation I according to this distribution π_T^*. In the remainder of this chapter, we will use this input language.

The small example given above can also be tried out in the online ProbLog inference engine: `https://dtai.cs.kuleuven.be/problog/editor.html`

4.3 Cinematographic Model

Whether they are movies, documentaries, lecture recordings, or any other form of edited video, all compositions tend to follow some generally accepted cinematographic rules. These rules exist to avoid confusing the viewer. While more experimental movies may deliberately violate these rules to shock or confuse the viewer, they are typically obeyed in the kind of objective event reports that form the focus of this chapter. Indeed, following these rules leads to informative, pleasant-to-watch reports, that do not confuse or distract the viewer.

Cinematographic rules can be divided in two main groups. The first group are rules that need to be taken into account while filming. These rules concern concepts such as depth of field, rule of thirds, headroom and the 180 degrees rule as described e.g., in [40]. These rules are outside the scope of this chapter, because they need to be taken into account by the virtual "camera men", not the virtual editor. We therefore assume that the raw video feeds satisfy these rules.

The second group of rules concerns the ways in which multiple video streams should be put together. These are the focus of our editing system. Before discussing these rules, we define some terminology. A *shot* is a sequence of successive frames from the same camera. A *cut* is a transition from one shot to the next. A *montage* is a sequence of shots. Figure 4.2 shows a montage composed of shots taken from different video streams. Note also that, at any particular point in time, some cameras may not be producing footage.

Length of shots A shot should neither be too long nor too short. If it is too long, the viewer will get bored, and his attention will drift. On the other hand, when shots are too short, the video becomes too jumpy, and the viewer is presented with too much information in too little time. In general, shot lengths between 7 and 90 seconds are considered acceptable. However, the length of a shot also depends on the amount of action in the scene: in energetic scenes, shorter shots can accentuate the ongoing action,

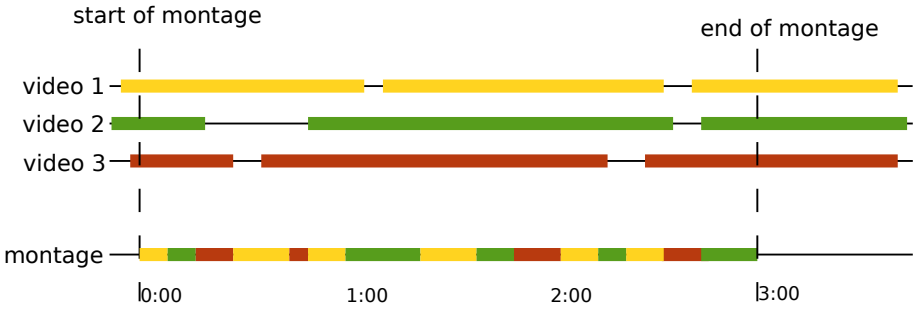


Figure 4.2: Making a montage out of video streams.

while longer shots are more suitable for static scenes. For instance, a lecture recording will tend to have longer shots than a recording of a basketball game.

In order to prevent the montage from appearing too mechanical or predictable, there should also be some variation in the shot length. Instead of striving towards some "ideal" shot length, we should therefore make use of different shot lengths, within the bounds of what is neither too long nor too short.

Order of shots Certain transitions between shots work well, while others should be avoided. When two consecutive shots mismatch, this is referred to as a *jump cut*. Jump cuts can occur when there is either too much or too little difference between the shots. When the footage of two different cameras is almost, but not completely the same, a "jumpy" effect occurs when switching from one to the other. As a rule, there should be a minimum angle of 30 degrees between the view points of two different cameras in order to avoid these kinds of jump cuts. In our case, this is guaranteed by the placement of cameras.

On the other hand, when the footage of two different cameras is completely different, the viewer will feel lost when switching from one to the other, because he has no clue about the context of the new subject.

A quite common order of shots is therefore to start with a *long shot*, that establishes an overview of the scene. The next shot is then typically a *medium shot*, that shows a closer picture of, e.g., a particular person in the scene. Then, a *close-up shot* of the person's face could follow. The close-up can then be followed up with another overview or medium shot. Although this order of shots is often pleasing, it is not necessarily always followed. Indeed, as with the shot length, there should also be some variation in the sequence of different kinds of shots in order to avoid montages that appear too mechanical.

Continuity To ensure coherence between shots, continuity should be respected. Cutting between disparate locations should be avoided, because it violates spatial continuity.When multiple cameras are filming one scene, all actions taking place in this scene should appear fluid and continuous. In other words, when switching between cameras, footage before and after the switch should be consecutive. This principle is called temporal continuity.

Our goal is to record footage of an event that takes place in a single location. We can therefore assume that spatial continuity is satisfied by our camera setup. In addition, our goal is also to cover the full length of the event (as opposed to summarizing the highlights). Therefore, temporal continuity will be satisfied because subsequent shots in the video will correspond to subsequent events in real life.

Action and reaction Video coverage of an event should capture all the relevant action. When filming an action, three separate phases are important: premeditation, the action itself and the reaction. First, whenever an action is about to take place, there is a brief moment in advance when the person is thinking about undertaking this action. To prepare the viewer for the action that is about to take place, showing this premeditation is important. Second, showing the action itself is of course the most important thing. Third, when the action is complete, the viewer expects to see the reaction of a person to it.

4.4 Overview of the Editing System

An overview of our editing system is shown in Figure 4.3. It takes as input a number of different video streams, together with an analysis of each of these streams. This analysis is performed by computer vision algorithms, which have been described elsewhere [40, 16, 17, 27, 39]. For each frame in each stream, we expect these algorithms to provide the following information:

- Whether people are present in the shot;
- Which kind of shot (long shot, medium shot, ...) it is;
- Which person is the most prominent subject of the shot;
- Which action (walking, talking, ...) the main subject performs.

The goal of our editing system is to decide for each point in time which of the available camera feeds will be used. The output of the system is the single video stream that is thus constructed.

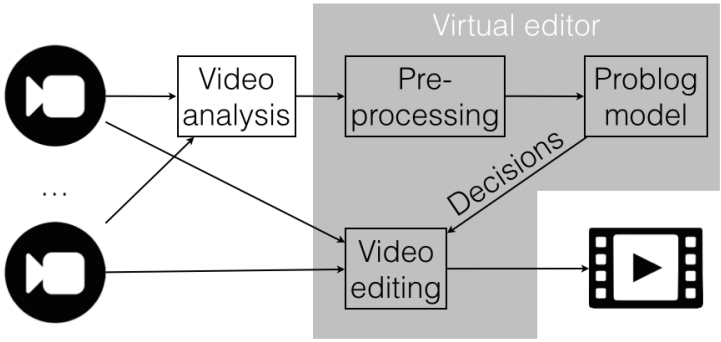


Figure 4.3: Overview of the editing system.

The editing system we propose consists of 3 components. The first component is the preprocessor, which comes in after the computer vision. This is a program written in python, which synchronizes camera footage, cleans up noisy detection data and parses this data to a format readable by ProbLog. Second, the ProbLog core works out a montage, taking into account the cinematographic rules described in Section 4.3. Finally, another python program edits the original video streams into one final montage, in the order the ProbLog core calculated.

4.4.1 Data Representation

We assume that all cameras are synchronized with respect to a global time line. This time line is divided into discrete frames. For each frame, we need to represent which kind of footage each of the available cameras is providing for that particular frame.

We represent camera sources as $camera(ID)$. When a new camera connects, or any active camera disconnects, it is possible to add or remove the corresponding ID. A connected camera does not necessarily produce useful footage at each point in time, e.g., a UAV does not produce footage while at its loading station. Valid video footage is represented as $frame(Frame, CameraID)$. Even when a camera is providing footage, this may not be usable if the camera is being adjusted. We denote this as $adjust(Frame, CameraID, Type)$ where $Type$ is either $zoom$ or $focus$.

Valid video footage may contain people or not. The video analysis system assigns each person—or rather, group of pixels that might be a person—a detection score that indicates how likely this is to be an actual person. We assume that the detection with the highest score corresponds to the most prominent person in the frame. We identify this person by means of a $person(Frame, CameraID, PersonID)$ fact. When a person is detected, we make a distinction between the different shot types: long shot,

medium shot or close-up. We represent this as $shot_type(Frame, CameraID, Type)$. In addition, the person may perform a specific action: stand still, walk, point, or talk. We represent this as $action(Frame, CameraID, Action)$. For images in which no person is detected, we make a distinction between $overview$ shots and $noperson$ shots. An $overview$ shot is one in which people are too small to be detected by the video analysis, whereas a $noperson$ shot is one that actually does not contain any people. The video analysis may use knowledge of the position of the different cameras in the scene to try to distinguish between the two.

4.4.2 Cinematographic Model

The editor system is based on a ProbLog model of the decision process that a human editor might follow to produce a live montage. At each point in time (i.e., for each frame), the editor needs to decide which camera to use, based on the footage that he has previously used, and the footage is currently available from the different cameras (and possibly also a small lookahead at future footage to avoid, e.g., switching to a camera that is about to stop producing footage).

For every time point T, the program decides which camera C to use at that time. This decision is recorded in a predicate $use_camera(T, C, T_{orig})$. The third argument records the starting time point T_{orig} of the shot that is ongoing at time T. This is merely for efficiency reasons, since we could also recompute T_{orig} by analysing all preceding $use_camera(T', C)$ atoms with $T' < T$.

We define $use_camera(T, C, T_{orig})$ by means of the following rules. The value of this predicate depends on two decisions that the editor must make: first, whether to cut away from the ongoing shot and, if so, which other camera to switch to. The first decision is recorded by the predicate $change(T)$ and we will discuss below how it is made. The effect of not changing is of course simply that the ongoing shot continues.

```
use_camera(T,C,Torig):-
    previous_camera(T,C,Torig),
    not(change(T)).
previous_camera(T,C,Torig):-
    time(T),Tp is T-1,use_camera(Tp,C,Torig).
```

If the editor does decide to change at time T, it will pick at random one of the other cameras C that are good candidates to switch to at that particular time (as given by $change_candidate(T, C)$).

```
use_camera(T, C, T):-
```

```
    time(T),change(T),
    findall(Cam,change_candidate(T,Cam),Cams),
    uniform(Cams,C,T).
```

The predicate $uniform(Cams, C, T)$ expresses that C is randomly selected, according to a uniform distribution, from the list $Cams$. The third argument T is present to ensure that, at different time points, a different camera may be selected from the same list. The definition of $uniform$ is as follows. In order to make a uniform selection from a list $[Head \mid Tail]$ of length N, this predicate either selects the head $Head$ with probability $\frac{1}{N}$, or it performs a uniform selection from the list $Tail$ of length $N-1$ with the remaining probability $1-\frac{1}{N}$.

```
uniform(List,El,ID):-
    length(List,L),uniform(List,L,El,ID).
uniform([Head|Tail],N,Head,ID):-
    P is 1/N,s(P,[Head|Tail],ID).
uniform([Head|Tail],N,E,ID):-
    P is 1/N,not(s(P,[Head|Tail],ID)),
    NN is N-1,uniform(Tail,NN,E,ID).
P::s(P,_,_).
```

The predicate $change(T)$ contains the essence of our cinematographic model. In general, there are two reasons to change: we can either do so because we want to, or because we have to. The latter case occurs when the current camera no longer provides usable footage, either because it no longer produces any footage at all, or because it starts to zoom or refocus.

```
change(T):-not(can_stay(T)).
can_stay(T):-
    previous_camera(T,C,_),
    frame_ok(T,C).
frame_ok(T,C):-frame(T,C),not(adjust(T,C,_)).
```

In addition to changing because we have to, we might also switch cameras because we want to. In general, this will occur if there is at least one camera whose footage we prefer over that of the current shot and if we have already met the minimal shot length requirement.

```
change(T):-
    change_candidate(T,_),not(too_short(T)).
```

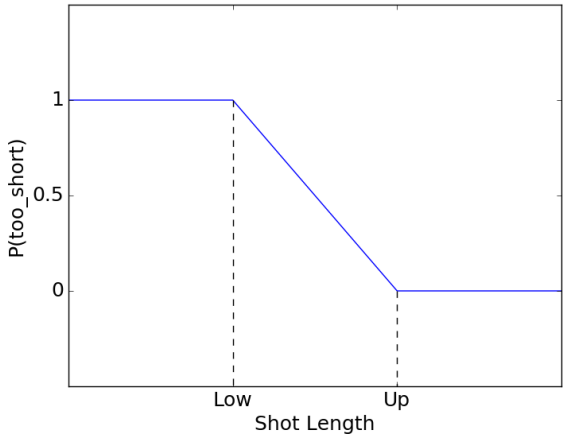


Figure 4.4: Probability of a shot being considered too short.

In order to decide whether the current shot is long enough, we of course need to know the current shot length. This is easily computed by means of the third argument of the use_cam predicate:

```
shot_length_until(T, Len):-
    previous_camera(T,C,Torig),Len is T-Torig.
```

The minimal shot length is not a fixed number. Some shots are definitely long enough, and some shots are definitely too short, but there is also a gray area, in which an editor might make a judgment call to cut away slightly sooner than he would like in order to, e.g., better capture the beginning of an action. For this reason, we define too_short in a probabilistic way, as shown in Figure 4.4.

Here, Up is an upper bound on the minimal shot length (i.e., longer than Up is never too short), while Low is a lower bound (i.e., shorter than Low is always too short).

```
P::too_short(T):-
    shot_length_until(T,Len),
    min_length_ub(Up),min_length_lb(Low),
    Len=<Up,P is min(1,(Up-Len)/(Up-Low)).
```

We similarly define a predicate too_long in terms of max_length_lb and max_length_ub. The effect of the current shot becoming too long is that this gives a reason to switch to a different camera:

```
P::too_long(T):-
    shot_length_until(T,Len),
    max_length_ub(Up),max_length_lb(Low),
    Len>=Low,P is min(1,(Len-Low)/(Up-Low)).
should_switch(T):-too_long(T).
```

We can now determine whether there exist cameras that it would be appropriate to switch to. Recall that if we find at least one such *change_candidate*, we will terminate the current shot unless it is still *too_short*. To make this decision, we divide the other cameras into three different categories: poor, fair and good change candidates. A good candidate is one that we always want to switch to, i.e., the existence of such a camera is in itself a reason to change. Poor and fair candidates are those that we might switch to if the current shot is getting too long and there is no better alternative available, i.e., we only consider switching to a fair candidate if there are no good candidates, and we only consider switching to a poor candidate if there are neither fair nor good candidates.

```
change_candidate(T,C):-
    change_candidate(T,C,good).
change_candidate(T, C):-
    change_candidate(T,C,fair),
    should_switch(T),
    not(change_candidate(T,_,good)).
change_candidate(T,C):-
    change_candidate(T,C,poor),
    should_switch(T),
    not(change_candidate(T,_,fair)),
    not(change_candidate(T,_,good)).
```

A poor candidate is any camera to which it is possible to change to. If, in addition, the candidate camera provides a good transition from the camera that is currently in use, it is considered a fair candidate. Finally, a good candidate is a fair candidate that also has more interesting footage than the current camera.

```
change_candidate(T,C,good):-
    change_candidate(T,C,fair),
    better_frame(T,C).
change_candidate(T,C,fair):-
    change_candidate(T,C,poor),
    good_transition(T,C).
change_candidate(T,C,poor):-can_change(T,C).
```

The predicate *can_change* is similar to the predicate *can_stay* that was defined above, but it is more stringent. In addition to demanding that the camera is currently providing

a usable frame ($frame_ok(T, C)$), can_change also demands that the camera will keep on providing usable frames in the near future. This is to prevent us from falling below the minimal shot length by cutting to a camera that is about to stop filming.

```
can_change(T,C1):-
    previous_camera(T,C,_),camera(C1),C\=C1,
    future_ok(T,C1).
future_ok(T,C):-
    camera(C),min_length_lb(M),T2 is T+M,
    not(between(T,T2,T1),not(frame_ok(T1,C))).
```

As discussed above, the difference between a poor candidate and a fair one lies in the quality of the transition that can be achieved by switching from the current camera to the candidate.

```
good_transition(T,C):-
    previous_camera(T,Cp,_),Tp is T-1,
    shot_type(Tp,Cp,STp),shot_type(T,C,STcur),
    shot_transition(T,STp,STcur).
```

Here, $shot_transition(T, ST_p, ST_{cur})$ represents the fact that, at time T, it is a good idea to switch from shot type ST_p to shot type ST_{cur}. As mentioned before, we do not want to impose a strict order in which the different shot types are used. For this reason, $shot_transition$ will be a probabilistic predicate. It will allow all possible transitions in principle, but assign a higher probability to those transitions that are generally considered better.

```
P::shot_transition(T,From,To):-
    time(T), quality(From,To,P).
quality(ls,ls,0.8).
quality(ls,ms,1).
...
```

The first argument of the $shot_transition$ predicate is needed to allow different choices to be made at different time points, i.e., it should be possible that a transition from ls to ms is made at time point t but not at $t' \neq t$. The $quality$ predicate defines the probability that each kind of transition is considered a good idea, as shown in Table 4.1.

The difference between a fair candidate and a good candidate is that the latter not only offers a good transition but also provides more interesting footage. This will mainly be determined by the actions that are taking place in the footage. Again, in order to

Table 4.1: Probabilities of different shot transitions.

	To				
From	ls	ms	cu	os	np
long **s**hot	0.8	1	0.2	0.8	0.1
medium **s**hot	1	1	0.6	0.8	0.1
close **u**p	0.8	0.7	0.2	0.8	0.1
overview **s**hot	0.8	0.7	0.1	0.8	0.1
no **p**erson	1	1	0.2	1	0.1

avoid giving the montage a mechanical feel, we do not place a strict priority on the different kinds of actions that might occur. Instead, we assign the different kinds of actions a probability that they will be selected in the video. Selecting an action can only happen at the time point when it starts. The video analysis module detects the actions of talking, walking and pointing. We also introduce a special action, called *emerging*, that we assign to a person who newly appears in the footage.

```
0.55::select_action(Taction,C,emerge):-
    start_person(Taction,C).
0.80::select_action(Taction,C,talk):-
     start_action(Taction,C,talk).
0.55::select_action(Taction,C,walk):-
    start_action(Taction,C,walk).
0.65::select_action(Taction,C,point):-
    start_action(Taction,C,point).
start_action(T,C,AT):-
    action(T,C,AT),Tprev is T-1,
    not(action(Tprev,C,AT)).
start_person(T,C):-
    person(T,C,ID),Tprev is T-1,
    not(person(Tprev,C,ID)).
```

As explained before, it is important to not just show the action itself, but also the "premeditation" leading up to it. The length of this lead-in may depend on the action in question. Moreover, as with shot lengths, there is a window of acceptable options, rather than a single fixed value. We define these windows for the different actions as follows:

```
pre_action_window(emerge,0, 1).
pre_action_window(talk,  2, 10).
```

```
pre_action_window(walk,  0, 2).
pre_action_window(point, 0, 5).
```

If an action AT starts at time T_{action} in the footage of camera C, then we can cut to camera C at any time point in the interval $[T_{action} - Max, T_{action} - Min]$, where $[Min, Max]$ is the pre-action window of action AT, as defined by the above predicate $pre_action_window(AT, Min, Max)$. For instance, if a *talk* action is initiated at time point 20, we can cut at any time point in the interval [10,18]. However, in doing so, we should take care that whenever we decide to switch at time T_{switch}, the camera actually produces a stream of valid frames between T_{switch} and T_{action}. For instance, if the camera produces no frames between time point 14 and 16, we should not yet switch to it at time point 12.

Starting from time point T, the following predicate gathers into a list all the time points $T' \leq T$ such that camera C has produced only valid frames between T' and T. We do not go back arbitrarily far in the history, but only look Len frames back.

```
valid_since(T,C,Len,[]):-not(frame_ok(T,C)).
valid_since(T,C,-1, []).
valid_since(T,C,Len,[T|Tail]):-
    Len>=0,frame_ok(T,C),
    Tp is T-1,Newlen is Len-1,
    valid_since(Tp,C,Newlen,Tail).
```

We can now use this predicate to gather into a $List$ all time points at which we could possibly switch to a camera C that shows action AT starting at T_{action}.

```
valid_action_window(Taction,C,List,AT):-
    pre_action_window(AT,Min,Max),
    T is Taction-Min,Delta is Max-Min,
    valid_since(T,C,Delta,List).
```

If we now switch to camera C at any time point $T_{switch} \in List$, we will have good footage from T_{switch} up to the time point T_{action} when action AT starts. In addition to this, we also want to avoid showing an action that does not last long enough for the viewer to make sense of it. We therefore also require that the person doing the action remains visible for a minimal amount of time after the start of the action.

```
action_visible(Taction,C,AT):-
    minimal_length(M),Tmin is Taction+M-1,
    not(between(Taction,Tmin,T),
        not(frame_ok(T,C),person(T,C,PID))).
```

If the starting action that we have selected (with $select_action$) satisfies both of these conditions (i.e., before and after the starting time of the action there is a sufficient amount of valid footage), then we can select one of its possible switching times (as given by the $valid_action_window$ predicate) as a time to cut to the camera in question.

```
better_frame(T,C):-
    select_action(TAct,C,Act),
    action_visible(TAct,C,Act),
    valid_action_window(TAct,C,List,Act),
    uniform(List,T,TAct).
```

In addition to switching to a different camera because it is starting an interesting new action, we might also switch because the footage of the current camera has become less interesting than the footage of another. We consider footage showing a person to be typically more interesting than footage not showing a person, and footage in which someone is talking more interesting than other footage. Moreover, some actions are more interesting in certain shot types, e.g., a person writing something on a blackboard can be best pictured in a medium shot, while it is better to picture a person that is pointing somewhere in a long shot, to actually see at what they are pointing at.

```
0.1::better_frame(T,C):-
    previous_camera(T,Cp,_),
    not(any_person(T,Cp)),any_person(T,C).
any_person(T,C):- person(T,C,_).
0.15::better_frame(T,C):-
    previous_camera(T,Cp,_),
    not(action(T,Cp,talk)),action(T,C,talk).
0.15::better_frame(T, C):-
    previous_camera(T, Cprev, _),
    action(T, C, write),
    shot_type(T, C, medium),
    not( action(T, Cprev, write),
        shot_type(T, Cprev, medium)).
0.08::better_frame(T, C):-
    previous_camera(T, Cprev, _),
    action(T, C, point),
    shot_type(T, C, long),
    not( action(T, Cprev, point),
        shot_type(T, Cprev, long)).
0.08::better_frame(T, C):-
    previous_camera(T, Cprev, _),
    action(T, C, gesture),
```

```
    shot_type(T, C, closeup),
    not( action(T, Cprev, gesture),
        shot_type(T, Cprev, closeup)).
```

This concludes the usual decision process of the editor. A special case that we have not yet discussed is the very first time point of the montage. Here, we prefer an overview shot. Failing that, we might choose a long shot, medium shot, close-up, or, as a final resort, a no-person shot. We record this preference in a list and gather the list of all possible candidates, taking these preferences into account. Then we select randomly one of the candidates.

```
initial_priority([os, ls, ms, cu, np]).
initial_candidate(T,C):-
    initial_priority(List),T2 is T+1,
    initial_candidate(T2, C, List).
initial_candidate(T, C, [First|Tail]):-
    shot_type(T,C,First),future_ok(T,C).
initial_candidate(T,C,[First,Second|Tail]):-
    not(shot_type(T,C,First),future_ok(T, C)),
    initial_candidate(T,C,[Second|Tail]).
use_camera(T,C,T):-initial_time_point(T),
    findall(Cam,initial_candidate(T,Cam),Cams),
    uniform(Cams,C,T).
```

4.5 Using the Cinematographic Model

The ProbLog model of the previous section describes the decision process that an editor might follow in order to produce a montage in accordance with cinematographic rules. This model defines a probability distribution over all possible ways in which a number of given input streams can be edited into a single video. It can be used to perform different tasks.

By *querying* the probability of a given montage according to this distribution, we can obtain an estimate of the quality of this montage: highly probable montages satisfy many of the rules, whereas unlikely montages contain many violations. *Sampling* from this distribution will produce a single montage. This is the main functionality we would like to achieve. Because the probability distribution defined by the ProbLog program assigns a higher probability to montages that respect more of the cinematographic rules, sampling is more likely to produce a good montage than a poor one. If we compute different samples for the same input, we will obtain a different montage each time. We view this as a desirable property, because it corresponds to how human editors perform

their task: it is unlikely that a human editor would produce precisely the same output each time, if he were asked to edit together the same input streams multiple times. Capturing this variance in our virtual editor helps to ensure that our montages have a natural feel.

An alternative to sampling is to compute the *most likely* montage, given a set of input streams. However, such an approach would have significant disadvantages when compared to the sampling approach:

- The task of computing the most likely outcome is computationally significantly harder than that of sampling from a distribution.
- To compute the most likely montage, we need to consider the entire time line at once. Such an approach would therefore only be usable in an offline editing system. By contrast, as we will discuss in Section 4.6, sampling can be used to implement a system that edits input streams in (slightly delayed) real-time.
- Under the same circumstances, the most likely montage will always make the same choices (namely, it will choose the most likely alternative). By contrast, sampling will occasionally make a less likely choice, thereby producing a less mechanical and more interesting montage.

One downside of the greater variation allowed by the sampling approach is that it may occasionally produce a poor montage. To compensate for this, our editing system will construct a set of samples. From this set, the most qualitative sample (i.e., the sample with the highest probability) will be returned as the output of our system. By making this set larger, we increase the computational cost of our method, but reduce the risk of returning a poor montage. Our current implementation uses 10 samples.

In addition to *querying* and *sampling*, the ProbLog system also contains algorithms that perform the task of *parameter learning*. Currently, the probabilistic parameters of our ProbLog model have been manually filled in, based on our understanding of cinematographic rules. An alternative approach would be to derive these parameters automatically from a number of examples videos. In this case, we might also be able to train our model to emulate particular editing styles. We investigate this topic in Section 4.10.

4.6 Achieving Real-Time Performance

In order to create a montage, our editing system calls on ProbLog to compute a number of different samples according to the probability distribution defined by the cinematographic model. There are two main parameters that affect the computational

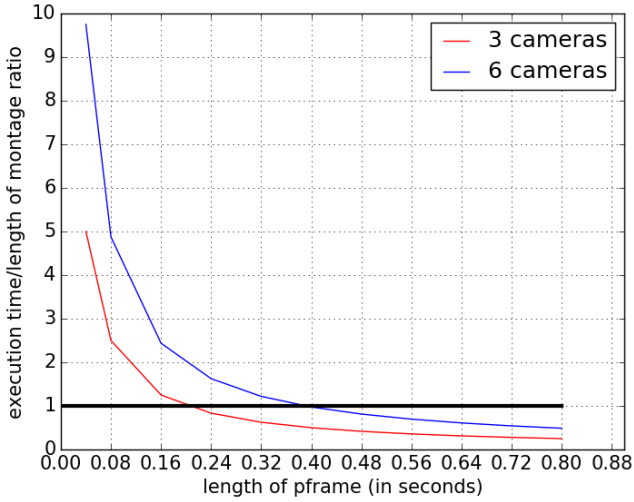


Figure 4.5: Execution time vs. length of p-frames (in seconds).

performance of the sampling algorithm: the number of time points and the number of cameras. Indeed, for each time point T and camera C, the sampling procedure must decide whether to switch to C at time T. We expect that the number of cameras will not vary greatly and will be dictated to a large extent by the setting in which the video must be produced. By contrast, the number of time points is a more flexible parameter. Indeed, the granularity of the time line used by our ProbLog model need not coincide with that of the actual input video. In other words, it is not necessary that each frame of the incoming video streams corresponds to an individual time point in the ProbLog model.

We will group together a number of actual frames into a sequence that we call a *p-frame*. These p-frames will act as time points for the ProbLog model. The number of real frames that go into one p-frame is therefore a key parameter of our system. On the one hand, this number affects the performance of the sampling algorithm: the higher it is, the faster the execution will be. On the other hand, this number also affects the quality of the montage that is produced, because our system will only be able to switch cameras at the start of a new p-frame. Therefore, the smaller the p-frames are, the more fine-grained this decision process will be and the higher the quality of the output.

Figure 4.5 shows the execution time needed to compute 10 samples in function of the length of a single p-frame. The y-axis expresses the execution time as a fraction of the length of the video stream that needs to be edited. If this value is ≤ 1, we have a system that is able to make the required editing decisions in real-time. The figure shows a

graph for both a 3-camera and a 6-camera setup. In both cases, taking a p-frame to be equal to a single video frame ($= 0.04s$, since the video was shot at $25fps$) produces a runtime that is much too slow. However, as the length of a p-frame increases, the runtime drastically decreases. For the 3-camera setup, we reach real-time performance as soon as at least 6 frames are combined in a single p-frame. With 6 cameras, we need 10 frames per p-frame. Because it is unlikely that a viewer would be able to tell the difference between cutting $0.4s$ earlier or later in a montage, we consider this an acceptable way of reaching real-time performance.

4.7 Discussion of Probabilities

This section discusses how the probabilistic nature of the model affects the resulting montage. We examine the accumulated probability of switching to a camera, the resulting quality of a montage and the effect of using p-frames. Because it is difficult to discuss these concepts when taking into account all influencing factors, we discuss each of these concepts in a local context. However, these concepts also occur when other factors come into play in a larger context.

4.7.1 Accumulated Probability

In the sampling algorithm discussed in Section 4.5, for each p-frame, an evaluation is performed to determine whether to change to a different camera or stay at the one that is currently used. There are a number of different conditions that might trigger a switch. If such a condition becomes true at some time point T_0, the probability to switch equals the stated probability P in the ProbLog theory. If that same trigger persists for an additional p-frame, the chance to switch at this subsequent frame T_1 is again P. The accumulated probability for switching for this reason then equals the sum of changing at T_0 (i.e., P) and the chance of changing at T_1, given that the switch did not occur at T_0 (i.e., $(1 - P) \times P$). When the trigger remains active for θ consecutive frames, the accumulated chance of switching to the corresponding camera can be calculated by the formula $1 - (1 - P)^{\theta+1}$. Figure 4.6 displays the accumulated probability of eventually switching to a camera considering the base probability and the number of frames the trigger persists. Notice that even triggers with a very small base probability, eventually have a high cumulative probability (i.e. $> 50\%$) to cause a switch.

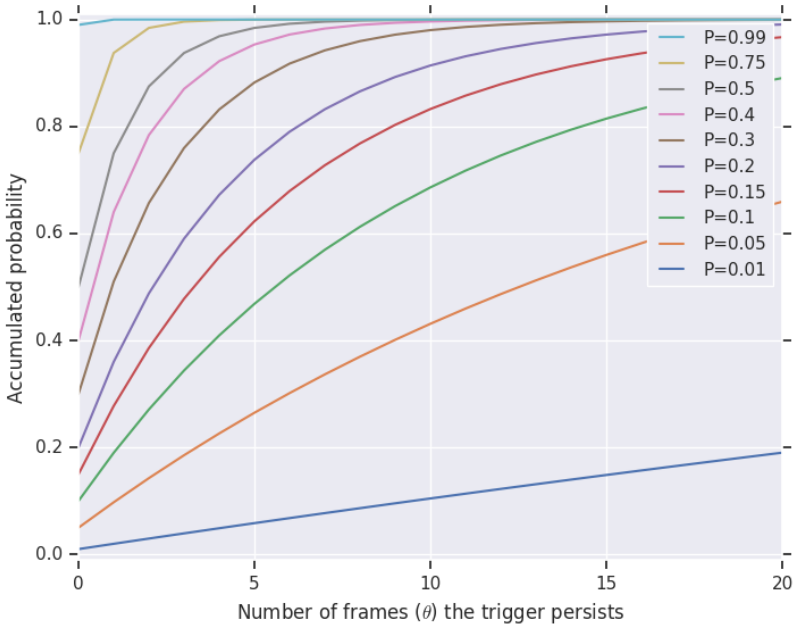


Figure 4.6: Accumulated probability of switching to a camera.

4.7.2 Quality of Montage

The fact that a switch is considered each p-frame not only affects the probability to switch to a certain camera, it also affects the total probability of the resulting model. This probability is used as a measure to evaluate the quality of the resulting montage, in order to rank the generated samples, as discussed in Section 4.5.

First, we consider the quality of all models where a switch is made. We vary the time θ since the initial occurrence of the trigger at p-frame T_0. When a switch occurs at T_0, the quality of the montage becomes P. Because the switch has been made, no decision has to be taken in the subsequent frames (because of the minimal shot length). As a result, the quality of the montage remains P for a while. On the other hand, when a trigger is ignored at T_0, but a switch is made at the next p-frame T_1, the quality of the montage becomes $(1 - P) \times P$. More generally, after θ frames, the quality of the resulting montage is $(1 - P)^\theta \times P$. As can be seen in Figure 4.7, regardless of the base probability, the quality of a montage decreases when delaying the switch. We can conclude that, if the system decides to switch, the quality of the montage is better when the switch occurs early. However, for triggers with a really low base probability, the

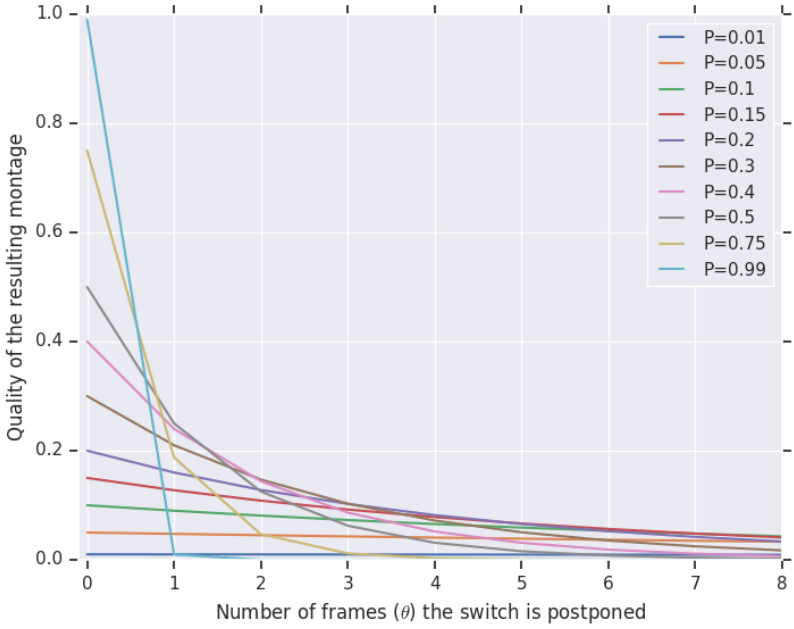


Figure 4.7: Quality of a montage where a switch is postponed θ frames.

decline in quality is negligible compared to the influence of other events.

Second, we consider the models where no switch is made, while a certain trigger persists for θ frames. The quality of the resulting montage is defined by the formula $(1 - P)^{\theta}$. As can be seen in Figure 4.8, the quality of a montage gradually decreases when a trigger persists, but no switch is made. This continues to the point that, eventually, the quality of the resulting montage is lower than the quality of a montage where a switch is made the moment the trigger initially occurs (T_0). We plot this threshold value in Figure 4.8, i.e., this Figure shows the threshold number of frames where switching at T_0 results in a better montage than waiting θ time frames. If the base probability is above 0.5, this quality is already higher at T_0. When the base probability is lower, the number of time frames before this threshold is reached gradually increases. For very small base probabilities, the threshold becomes unrealistically high, and the quality of a montage where a switch occurs will rarely out-qualify that of not switching. This supports the intuition that events with very low probabilities result in a low quality montage.

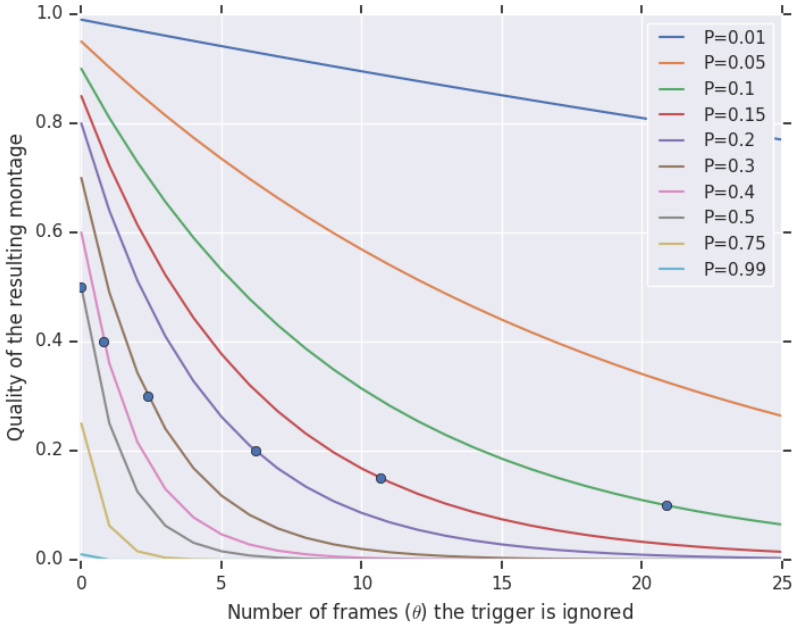


Figure 4.8: Quality comparison of switching at T_0 and ignoring a trigger for θ frames.

4.7.3 Effect of P-frames

It is obvious that the use of p-frames affects both accumulated probability and the quality of the montage.

Since choices are made less frequently, the accumulated probability per second decreases significantly for the same base probability P. However, it is possible to raise the base probability, to counter this effect. To obtain the same accumulated probability over time, the probability in the p-frame theory can be calculated as $P_p = 1-(1-P_f)^R$. Where P_f is the probability in "regular" frames, P_p is the probability in p-frames, and R is the ratio $\frac{frames}{pframe}$.

An additional consequence of using p-frames lies in the quality of the montage. First, because choices are made less frequently, the overall probability of a montage will be considerably higher. However this does not have an impact on the quality ranking of the resulting montages, because this affects all samples equally. Second, the turning points in Figure 4.8 are affected by the use of p-frames, even when adjusting the probability. To obtain the same turning points in time, the probabilities should satisfy

$\frac{ln(P_p)}{ln(1-P_p)} = R\frac{ln(P_f)}{ln(1-P_f)}$.

Both the accumulated probability and the resulting quality are affected by the ratio $\frac{frames}{pframe}$. In other words, a change in the p-frame length requires adjustments to the probabilities in the model.

4.7.4 Discussion

To recapitulate the last sections, we compare three categories of triggers: those with a probability $P > 0.5$, those where $0.1 \leq P \leq 0.5$ and those where $P < 0.1$. Because the length of p-frames affects the probability values in the model, we discuss the probability values we used for a p-frame rate of 2.5 p-frames per second. Similar conclusions can be drawn for different p-frame rates, albeit with slightly altered probability values.

Triggers that have $P \geq 0.5$ have a fairly high chance of switching instantly. Moreover, the quality of switching immediately is inherently higher than that of postponing the switch or not switching at all.

For persistent triggers that have $P < 0.5$, initially it might seem like not switching seems a better choice (i.e. both the probability and quality of switching is lower than that of staying). However, on the one side, when the trigger persists, the quality of the montage where no switch is made falls below the quality of the montage where the switch was performed directly. On the other hand, the chance of switching due to this trigger accumulates to a fairly high probability over time.

When a trigger has $P << 0.1$, the quality of a montage where no switch occurs generally exceeds that of one where a switch occurs. Moreover, the accumulated probability of switching remains fairly low. These triggers will rarely activate and are often outweighed by other triggers. Only in the rare occasion no single other reason triggers any switch, i.e., nothing interesting happens in any camera, such a trigger ensures a minimal shot transition.

4.8 Experimental Results

Our system is able to produce real-time edits of different video streams. The only question remaining is whether the resulting montages are of a quality similar to those produced by professional editors. Since there is no single right way to edit video, we have no "ground truth" to compare the output of our system to. Instead, we have subjected the virtual editor to a "Turing test": we have asked a number of test subjects

to distinguish between the output of our system and a professionally made montage of the same video streams.

The test case used in this experiment is a lecture recording made by three cameras. This footage was edited by a professional editor, who was present during the recording. From the entire video stream, we selected a fragment of three minutes in which there was a lot of "action", namely one speaker introducing another speaker, who then came to the stage. We presented 58 students with two video clips: this particular fragment as edited by the professional, and the same fragment edited from the same input streams by our system. The student were then asked to identify the clip produced by the professional. As an incentive, a small prize (worth around 75€) was given to a random student among those who guessed correctly.

The two video clips can be viewed online:

1. `https://www.youtube.com/watch?v=7vrfhzD4G0c`
2. `https://www.youtube.com/watch?v=Bzl10YKGeI4`

In case the reader would like to perform this experiment himself: the professionally edited video is Montage i, where i is the 22^{nd} digit in the decimal expansion of π.

Of the 58 students, 31 ($= 53\%$) correctly identified the professionally edited clip. This difference between this outcome and one that could be produced by random guessing is not statistically significant ($T[57] = 0.5$; $p = 0.6$), i.e., the data does not allow to reject the hypothesis that the subjects were unable to tell the difference between the professional editor and our system. In addition, the subjects were also asked to indicate (on a scale of 1 to 3) how confident they were in their choice. The results are shown in Figure 4.9. Those students who guessed *in*correctly were on average actually slightly *more* confident of their choice than those who guessed correctly, although again the difference is not statistically significant (a $\chi^2[2] = 1.69$ test provides a p-value of 43%).

This experiment demonstrates that our editing system indeed provides a good approximation of the quality delivered by a professional editor for this particular case study of lecture recording.

4.9 Readable Representation

The ProbLog syntax is not readable enough for domain experts to comprehend the represented knowledge. We therefore explore the possibility to express the knowledge in this model in the cDMN framework proposed in Chapter 2. This framework, however is not designed express probabilistic knowledge.

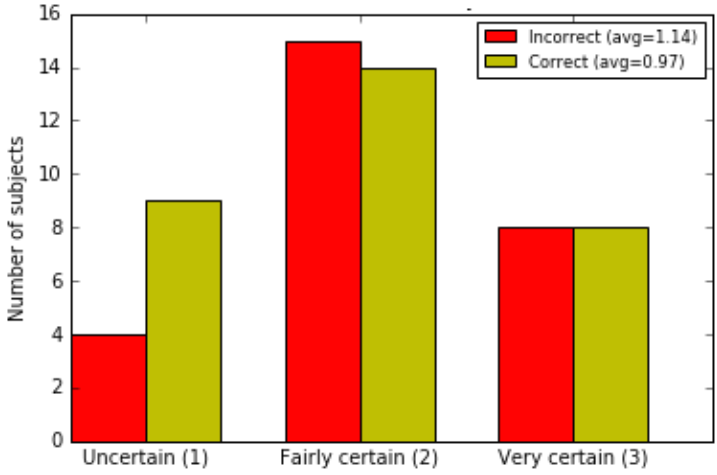


Figure 4.9: Confidence of subjects in their choice.

In order to be able to represent the cinematographic knowledge, we slightly alter the semantics of predicates. We define the semantics of a decision table with a single output header expression $Probability\ of\ H_j$ (where H_j is a boolean or a predicate), and which is assigned a probability $P_i \leq 1$, as the set of CP-logic rules:

$$P_i :: H_j \leftarrow \bigwedge_{j \in I} F_{ij}(H_j)$$

Where i is a row of the table, and I is the set of input columns.

Using these semantics, the first line of the "Good Transition" decision table in Figure 4.10 corresponds to the CP-logic formula:

$$\begin{aligned}
&\forall T[Time], Old[Camera], Candidate[Camera] \\
&\quad (0.8 :: goodTransition(Candidate, T)) \leftarrow \\
&\quad\quad Old = cameraUsed(T-1) \wedge \\
&\quad\quad shotType(Old, T-1) = overview \wedge \\
&\quad\quad shotType(Candidate, T) = overview.
\end{aligned}$$

The representation proposed in this section is merely meant as a proof of concept. Currently, no automatic conversion is supported for this syntax to either CP-logic, FO(·) or ProbLog.

Good Transition

A	**Time called T**	**Camera called *old cam***	**Camera called *candidate***	**Shottype in *old cam* at *T*-1**	**Shottype in *candidate* at *T***	**Probability of Good transition to *candidate* at *T***
1	–	Camera at *T*-1	–	overview	overview	0.8
2	–	Camera at *T*-1	–	overview	medium	0.7
3	–	Camera at *T*-1	–	overview	overview	0.1
…	…	…	…	…	…	…

Better shot

A	**Time called *T***	**Camera called *old cam***	**Camera called *candidate***	**Action**	**Action in *old cam* at *T***	**Action in *candidate* at *T***	**Shottype in *candidate* at *T***	**Probability of Better shot of *candidate* at *T***
1	–	Camera at T-1	–	Person	No	yes	–	0.1
2	–	Camera at T-1	–	Talk	No	Yes	–	0.15
3	–	Camera at T-1	–	Write	No	yes	Medium	0.15
…	…	…	…	…	…	…	…	…

Quality of Candidate

A	**Time called *T***	**Camera called *candidate***	**Can change to *candidate* at *T***	**Good transition to *candidate* at *T***	**Better shot of *candidate* at *T***	**Quality of *candidate* at *T***
1	–	not(Camera at *T*-1)	Yes	No	–	Poor
2	–	not(Camera at *T*-1)	Yes	Yes	No	Fair
3	–	not(Camera at *T*-1)	Yes	Yes	Yes	Good

Next Camera

A	**Time**	**Switch at *Time***	**Camera at *Time***
1	–	No	Camera at *Time*-1
2	–	Yes	One of best candidates at Time

Figure 4.10: Extract of the decision logic of montage knowledge in cDMN.

Figure 4.10 displays an extract of the knowledge in the cDMN format. We briefly discuss the logic in these tables. The *Good Transition* decision table defines whether a transition of a certain shot type to another is appealing. E.g., it states that the probability of a transition from an overview shot to another overview shot being appealing is 80%. The logic in this table is equivalent to that in Table 4.1 and the corresponding `good_transition` ProbLog predicate.

Analogously, the *Better shot* table defines when the footage of a candidate camera is better than the currently used footage. For example, the first line in this table defines that at each point in time, a candidate has a 10% probability of having better footage than the previously used one, if the candidate shows a person and the previously used one does not.

Type		
Name	**Type**	**Values**
Time	int	[1, 1000]
Camera	string	cam1, cam2, cam3, cam4
Quality	string	Good, Fair, Poor
Shottype	string	overview, long, medium, close up
Action	string	Person, Talk, Write, ...

Relation
Name
Switch at Time
Good transition to Camera at Time
Better shot of Camera at Time
Action in Camera at Time

Functions		
Name	**Type**	**Values**
Camera used of Time	Camera	Camera at Time
Quality of Camera and Time	Quality	Quality of Camera at Time
Shottype of Camera and Time	Shottype	Shottype of Camera at Time
One of Best Candidates of time	Camera	One of Best Candidates at Time

Figure 4.11: Extract of the cDMN glossary of the montage knowledge.

The table *Quality of Candidate* determines the quality of a certain candidate, based on a number of factors. When it is possible to change to a candidate camera (different from the one previously used), but there is no good transition, this is a `poor` candidate. Analogously, a `Fair` candidate is defined as a camera whose current footage would provide a good shot transition, but which does not have a better shot (i.e., more interesting footage) at that time. Finally, a candidate is considered as `Good` when it shows more interesting footage and has a pleasing shot transition. This table represents the `change_candidate` predicate in Section 4.4.2.

Finally, the *Next Camera* table specifies which camera should be selected at each time. When no switch should be made, the camera at that time stays the same as the previously used camera. When a switch should be made, on the other hand, the Camera should be selected from the candidates with the highest quality. For this purpose we introduce a function `One Of Best Candidates at Time`. This function should randomly select one of the camera's with the highest quality at time T, according to a uniform probability distribution. However, the cDMN notation can currently not be used to express such construct. It would, however, be interesting to investigate this in future work.

An extract of the cDMN glossary for this problem can be found in Figure 4.11. As can be seen, this glossary has the standard cDMN glossary syntax.

We can conclude that, even though the cDMN notation has not been specifically designed to represent this kind of non-deterministic knowledge, it is able to elegantly represent most parts of this knowledge. Some constructs (such as sampling according to a uniform distribution), however, can currently not be expressed in this notation.

4.10 Parameter Learning

We initialized the probabilities in the system, based on the know-how of a cinematographic expert. However, it would be interesting to learn these probabilities from gathered data. Problog offers the functionality to learn parameters of a model, by providing a number of interpretations of that model. In the context of video montage, this means that a number of examples should be provided which contain the needed input (raw streams and their annotations) and the respective output (a desired montage). Using these examples, ProbLog then learns the probabilities of each of the switching criteria. This would enable us to learn different models for different types of settings (e.g. lecture recordings, sports events).

Unfortunately, this kind of data is not commonly available. Either the raw videos are available, but no professionally edited resulting montage, or the resulting video is available, but the original raw videos are no longer accessible. We therefore generate synthetic data to experiment with the possibility to learn these probabilities.

The learning step requires two major components: the model, and a number of examples. We discuss each of them separately.

Simplified Model We want to learn the probabilities of each of the switching criteria in the model outlined in Section 4.4.2. However, this model contains a substantial amount of knowledge. Learning the parameters of the full model at once is infeasible because of computational complexity. We therefore reduce the model's complexity, without deviating too much from the original. Because the learning starts from specific models, in which choices have been made, some predicates become obsolete. We make a number of simplifications to the cinematographic model.

- It is no longer necessary to determine whether `future_ok` holds.
- The complex probability distribution for the `too_short` and `too_long` predicate can be simplified to a static probability.
- Moreover, it is possible to simplify uniform sampling in e.g. `use_cam` to a static probability.
- `can_change` is obsolete because it inevitably holds if a switch occurred.
- `action_visible` and `valid_since` are no longer needed.
- the mechanism to select an `initial_candidate` (including the `initial_priority` and `lower_time_boundary` predicates) can be dropped.

- We assume that all `shot_transition` are pleasing, which greatly reduces complexity of the model.

The simplified model then consists of all reasons to switch, i.e., the `too_long` predicate, the `can_stay` predicate, and the `better_shot` predicate, with all their needed help predicates, such as `shot_length_until`, `select_action` and `pre_action_window`. Moreover, it includes the `change_candidate`, `change` and `should_switch` predicates, as well as the key predicate `use_camera`. Finally, the model consists of a number of leak probabilities for the input predicates (e.g. `input_person` and `input_action`) and the `use_camera` at t_0.

The goal of this learning step is to determine each of the probabilities of the `better_shot` predicate, i.e. to learn how these different reasons define the learning examples.

Generating Examples We generate a number of examples with the procedure outlined in Section 4.5. Each example contains evidence that contains both the input that was used to generate the montage (i.e., the availability of cameras, the presence of people in the frame, etc.) and the generated montage (i.e., when each camera is selected). In order to reduce the computational complexity, we limit the number of frames per example to an absolute minimum. Each example consists only of the used camera at the current time and the previous time.

Using this approach we generated 5000 examples, which could be extended unlimitedly if needed.

Experimental Results Despite the simplification of the model and the limited size of the examples, ProbLog is not able to learn the model's parameters because of a memory overflow (the used system has a memory size of 32 Gigabytes).

As a final effort to reduce the complexity, we fix all but one probability, reduce the training set to one single example and repeat the experiment. Of course this would yield no directly useable results, because the learned probability is dependent on all (ad-hoc selected) other probabilities in the model and is based only on one time step. Even in this most basic case, ProbLog is unable to determine the one to-be-learned probability in the model.

We have to conclude that the ProbLog parameter learning functionality cannot be used in our cinematographic model.

4.11 Related Work

There exist several systems that perform automatic video montage for the purpose of creating a summary of some event(s). Examples in the scientific literature are [32, 34]. Also commercially available software offers this functionality, e.g., Muvee[2], Aescripts[3], Magisto[4] and Google Photos[5]. While related, this is essentially a different problem from the one that we have considered in this chapter, where we want to produce full-length coverage of an event.

This problem of producing full-length coverage is also studied by [43]. Their approach also takes into account cinematographic rules regarding shot transitions and view selection. However, it requires about one second of computation time per frame and does not include an experimental validation of the quality of the produced video. In [57], a general scheduling algorithm is proposed to achieve optimal observability using multiple adjustable sensors. This algorithm is applied to the video editing problem in [19]. It requires cameras with (partially) overlapping field of views, and constructs a 3D-map with areas of higher interest. Quality of view is determined by the amount of action, number of objects, visible events and a combined object score. The video montage is then made by maximizing this quality, while simultaneously minimizing inter-camera switching. This is a significantly different approach from ours, in which the cinematographic rules are not as explicitly present. The quality of videos created by this system was compared to that of professionally edited videos in an experiment similar to ours: 83% of their test subjects labeled the computer generated montage as professionally edited, while 93% of the test subjects labeled the professionally edited video as such. While this is not precisely the same experiment as we performed, our algorithm seems to achieve similar results. In addition, this method also requires $0.16s$ of decision-making time per frame, which does not suffice to reach real-time performance at 25fps.

Both of these methods reduce the video editing problem to an optimization problem, which explains their greater computational complexity.

In [46], an automatic video editor is developed specifically for lecture recording. Three cameras are used, each with a specific purpose: an overview camera, an audience facing camera and a lecturer tracker. This work includes cinematographic rules that are specific to this setting (e.g., if a person in the audience asks a question, show that person in the montage). The quality of the produced montages was tested by an experiment in which subjects were asked to assign a score of 1 to 5 to both an automatically produced and professionally made montage. The automatically generated video scored an average

[2] http://www.muvee.com/home
[3] http://aescripts.com/automated-video-editing
[4] https://www.magisto.com/how-it-works
[5] https://photos.google.com

of 2.8, while man-made video had an average score of 3.7. This difference of 22.5% strikes us as more significant than the differences we observed in our experiments. This system was later extended to allow a greater variety of settings [58]. This work also added some additional rules to the system, but their impact on the quality of the montages was not experimentally verified.

Another setting-specific system is that of [33], which performs offline automatic video editing of a theatre play. This system uses a setup with one static camera, from which a range of sub-views are cut to create multiple shots. The quality of the montages produced by this system was not experimentally verified.

4.12 Conclusions and Future Work

In this chapter we showed that the domain of knowledge representation can be used to express non-deterministic knowledge. We present an automated video editing system, which may play a role in reducing the cost of producing a video report of an event such as a lecture, sports game or musical performance. We have developed this system in a declarative way, by building a model of the non-deterministic decision process that a human editor could follow in order to produce a montage. By using the state-of-the-art Probabilistic Logic Programming system ProbLog, we are able to sample from this distribution and thereby produce a montage. We have demonstrated that the computations needed to make all the required decisions can be performed in real-time and that—at least for the particular case study of lecture recording—the quality of the produced montages is comparable to that of montages produced by a professional editor. Moreover, we showed that even non-deterministic knowledge can be represented in a readable format. Unfortunately, even with numerous simplifications to the underlying model, we did not succeed in using the ProbLog learning functionality to use data to improve the constructed knowledge model.

In future work, it would be interesting to investigate other machine learning methodologies to fine-tune the probabilistic parameters of the proposed declarative model. Using this methodology would enable us to learn a specific model for specific kinds of events. These updated models could then be used to examine the performance of the proposed system in more complex settings than lecture recordings.

Moreover, another interesting topic to further investigate is how cDMN can be used to represent probabilistic logic, and which inference tasks could be offered on the resulting models.

Chapter 5

Alarm Management on a Liquid Bulk Terminal

In this chapter, we study the alarm management in a liquid bulk terminal in the port of Antwerp, Belgium.

The research in this chapter differs from that in the previous chapters in the sense that, so far, the used algorithms were all built starting from domain knowledge provided by domain experts, whereas the use case in this chapter does not provide sufficient domain knowledge to construct such a model. We therefore base the model solely on knowledge that can be extracted from data. However, our goal remains to develop an interpretable automatic system that can support domain experts. Therefore, we slightly alter our approach, as can be seen in Figure 5.1 We use the functionality of an ML technique to construct an ML model that is able to predict the outcome of the system. Then we present the knowledge in this model to the domain experts in such a way that they can interpret it. This approach produces a readable model of the domain knowledge that is present in the data.

The research in this chapter is funded by the Flemish agency for Innovation (VLAIO) as part of the research project E'XperT, which was conducted in collaboration with the companies Agidens and OTSA. The content of this chapter is based on joined work with Kylian Van Dessel, and has been presented and published at KI 2017 [6].

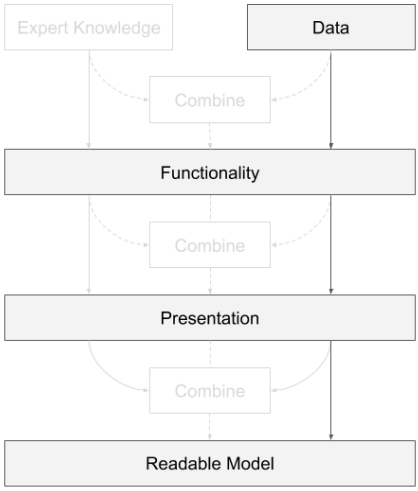


Figure 5.1: Visualization of the research topics in this chapter.

5.1 Introduction

This chapter describes work conducted in collaboration the company *Agidens* on alarm management for a liquid bulk terminal in the port of Antwerp, Belgium. This terminal handles various fluids and gasses. These typically arrive by ship, are transferred to on-site tanks, and are later transferred again to trucks, trains or barges. All of this requires a large and complex network of pipelines, valves, tanks and pumps.

The terminal is monitored by operators in a control room, who ensure that scheduled tasks are executed as planned. E.g., when a new transfer needs to be started, the operators are responsible for establishing a route between the two endpoints (e.g., from tank to loading station) by opening and closing the appropriate valves.

Because of the dangerous products, safety is very important. The installation contains numerous sensors that raise alarms as soon as parameters leave the expected operating conditions. Handling these alarms is a significant part of the workload of the control room operators. Moreover, significant expertise is required to distinguish important alarms from irrelevant ones.

The main goal of this project was to investigate the use of Artificial Intelligence technology to improve the alarm handling workload of the operators. Before going into more detail, we discuss some relevant properties of the terminal.

Discontinuous operations. Activity on the terminal consists of different processes,

which are typically rather short-lived, lasting at most a few hours. Moreover, each of these processes (i.e., transfer operations between two end-points) is almost unique, in the sense that even when they share a common origin and destination, the route between them may be different, as may the product that is being transferred, its temperature or its level in the tank.

Changing circumstances. The terminal is continuously undergoing maintenance and expansion. From one day to the next, the layout of the terminal may therefore be different. Moreover, normal operations are sometimes halted in order to perform these tasks, which may itself cause a flurry of alarms.

No complete terminal model. Information about the terminal is spread out over different databases. During the project, we investigated whether it would be possible to integrate these databases into a single coherent model. However, the companies involved decided against this, because of the prohibitive cost of constructing such a model and of keeping it up-to-date.

Infrequent and diverse alarms. Roughly 20 000 different alarms can be generated in the terminal. Obviously, many of these occur very infrequently.

5.2 Related Work

It is widely recognized that, in a complex modern industrial environment, the task of operators is growing ever more complex, and that technological support is needed to keep this manageable [28]. Several opportunities exist for Artificial Intelligence technology to play a role in solving this problem.

Many approaches focus on providing real-time advice to the operator about a specific alarm, e.g., by means of a root cause analysis [67]. This kind of functionality typically requires detailed knowledge about the layout of the industrial system that is being monitored [60]. Because, in our case, information about the layout of the terminal is not available, these methods are not applicable for us.

A different approach is to reduce the number of alarms shown to the operators, e.g., by filtering them according to specific rules [9, 70, 35]. In the course of this project, we extensively discussed this possibility with the operators. However, because of the dangerous nature of the chemicals handled in the terminal, no sufficiently safe policy for hiding alarms could be determined.

5.3 Project goals

There are several ways in which the workload of operators might be reduced. Initially, the we considered providing online advice to the operators, to help them handle alarms more quickly. However, this approach was abandoned for a number of reasons. First, in order to give specific advice for a concrete situation, the present condition of the terminal needs to be known. While the status of individual components is known, the lack of a complete model of the terminal means that it is unknown how these components relate to each other. Because of this lack of knowledge, it would be difficult to achieve a high degree of accuracy. Second, a closer inspection revealed that the operators typically do not spend a lot of time handling individual alarms. Since their main focus is on completing the tasks at hand, they typically simply retry the operation, reroute around a damaged component and/or call maintenance. To significantly affect their workload, it is more important to reduce the *number* of alarms that occur in the first place, rather than the *time* spent on each alarm. Third, because of the dangerous products being handled, it is not desirable to actually hide alarms from the operators: the final decision should always be in the hands of a human operator.

For these reasons, we decided not to focus directly on the operators themselves, but to focus instead on the terminal's process engineers. By helping them to better understand which alarms are occurring and why, we can enable these engineers to improve the processes on the terminal, eliminate the underlying cause of alarms, and thereby indirectly also reduce the number of alarms that the operators have to deal with.

5.4 Identifying Interesting Alarms

As a first step towards helping the process engineer reduce the number of alarms, we attempt to identify alarms that are interesting for him to take a closer look at. Discussions with the engineers revealed that two parameters are important for this. First, of course the number of occurrences of an alarm is important: analysing alarms that occur very rarely is probably not a good investment of the engineer's time. Second, our system can create the most added value for the engineers by alerting them to alarms that are lingering in the background. An alarm that occurs repeatedly during a single shift will be reported by the operator and so the process engineer will already be aware of it. However, an alarm that occurs, e.g., only a few times each week, might go unnoticed.

Based on these two parameters, we propose the visualization shown in Figure 5.2. In this scatter plot, the Y-axis plots the number of occurrences of the alarm. The X-axis attempts to capture the degree to which this is a "lingering alarm". To this end, we fit a Poisson distribution to the occurrence graph of the alarm. This distribution expresses

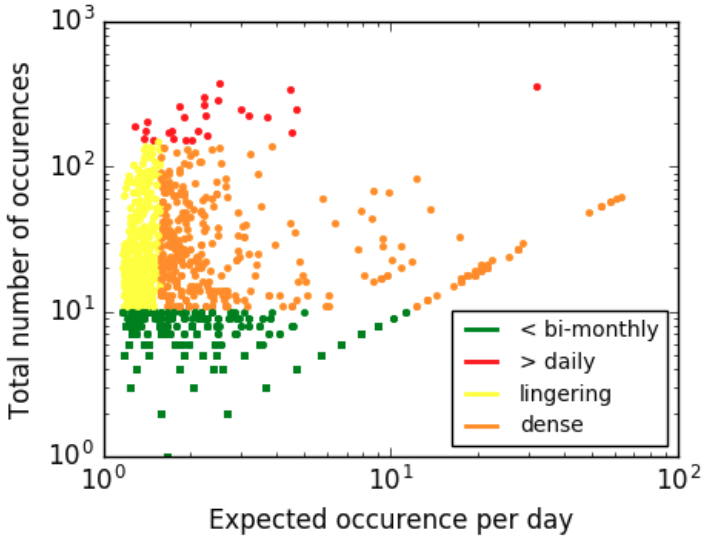
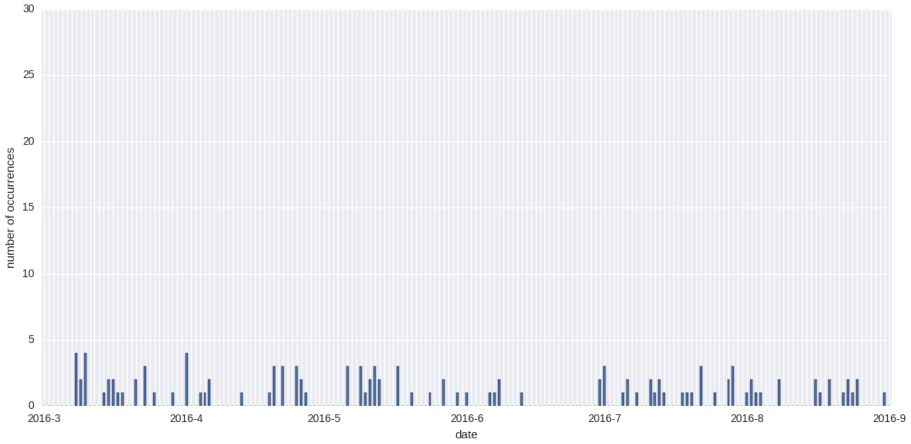


Figure 5.2: Visualization of alarm occurrences.

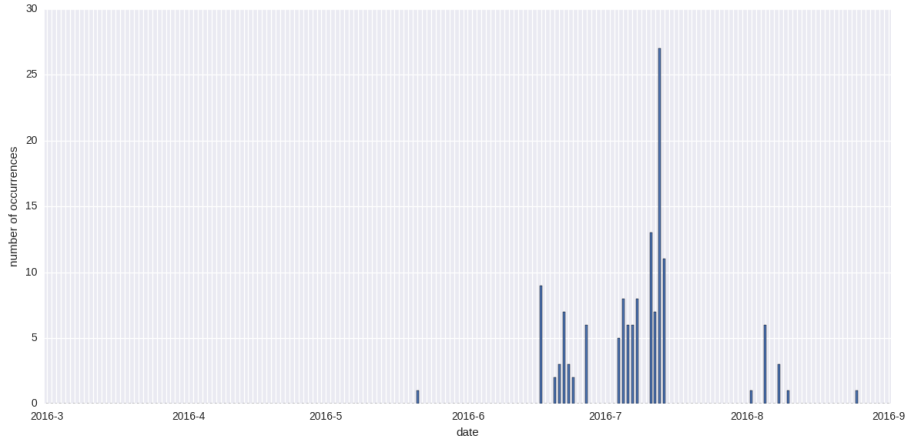
the probability that a certain event (an alarm) occurs a specific number of times in a fixed time period (a day). It is characterized by one parameter, λ, which represents the expected number of occurrences of such alarm in a single day. Events that occur only sparsely have a low λ parameter, a higher λ symbolize events that occur more frequently. Figure 5.3 shows an example of an alarm for which the parameter λ of the best-fit Poisson distribution is low and one for which it is high. Those with low λ are the kind of "lingering" alarms in which the process engineers are interested.

To determine this parameter λ, the following procedure is used. First, we generate a histogram that counts the number of days an alarm has occurred k times that day. A Poisson distribution is fitted to this histogram, i.e., $P_{Poisson} = \frac{\lambda^k e^{-\lambda}}{k!}$.

Because alarms are sparse events, there are significantly more days in which an does not occur than there are days in which it does. This phenomenon confuses the algorithm that attempts to fit a Poisson distribution to the data. A solution is proposed by Lambert [45], which is called a Zero-Inflated Poisson (ZIP) model. This algorithm assumes that, in addition to the Poisson distribution, there is a second underlying process that determines whether an alarm does or does not occur on a specific day. This model is characterized by an additional parameter π that captures the amount of excess



(a) Typical alarm with a low λ (lingering alarm).



(b) Typical alarm with a high λ (high density alarm).

Figure 5.3: Comparison of alarms with a low and high Poisson λ .

zeros:

$$P_{ZIP}(k = 0) = \pi + (1 - \pi)P_{Poisson}$$

$$P_{ZIP}(k > 0) = (1 - \pi)P_{Poisson}$$

Even though we are not interested in this excess number of alarm-less days π, this ZIP model better fits the data. Therefore it enables us to make a better distinction between lingering alarms with a low λ parameter, and other alarms with a higher λ parameter.

When we originally published our results in [6], a regular Poisson distribution was used instead of a Zero-Inflated distribution. At the time, this also seemed to fit the data quite well. However, this was due to an unfortunate error in our implementation. Correcting the error revealed the need for the ZIP model.

We define four different areas in the graph of Figure 5.2:

1. Alarms occurring less than twice a month (green).
2. Alarms occurring more than once a day (red).
3. Other alarms with λ parameter below median, i.e., "lingering" alarms (yellow).
4. Other alarms with λ parameter above median (orange).

Figure 5.4 shows a screenshot of the user interface of our system. It allows the user to generate and compare graphs based on a number of filtering options.

The panels at the left and right side of the interface provide a number of filter options which apply to the top-left and top-right graph, respectively. There are filters on alarm *Category*, *subsystem* (e.g. acetate), *time of day* (day is from 6 a.m. till 6 p.m. and night is the opposite), *component type* (e.g. temperature sensor) and *time period*. In addition to filtering the alarms, they can also be grouped by one of the following parameters: *Message* (default), *id, class, category, subsystem, time of day* or *type*.

The configuration in Figure 5.4 compares alarms in the acetate system of the last month (on the right) to the alarms of a period of two months before that (on the left). The bottom graph shows a comparison of the top-left and top-right graph. For every alarm that appears in both top graphs, the difference (on both axes) between the top-right and top-left graph is plotted in the bottom graph. To get an accurate picture of the difference in occurrences, the Y-values are first scaled to the average number of occurrences per month.

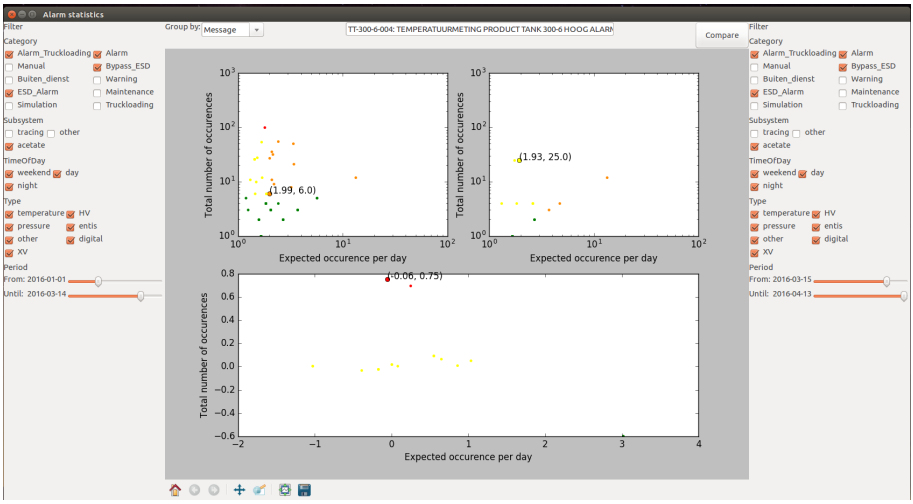


Figure 5.4: Proposed graphical interface.

5.5 Automated Analysis of Alarms

Once the process engineer has found an alarm to investigate further, our system will attempt to help him identify the circumstances in which this alarm occurs. It does this by training a machine learning model to predict the occurrence of this alarm. However, our main focus is not on achieving a high prediction accuracy, but rather on constructing models that can easily be interpreted by the engineer. For this reason, we have chosen to use a decision tree learning algorithm.

On a side note, accuracy is a poor measure to evaluate the performance of predictors of highly skewed datasets, because it does not take into account the imbalance between classes. The performance of a predictor can better be calculated by means of its precision and recall value. We are mostly interested in achieving a high precision value (i.e., a predictor that avoids non-alarms being predicted as alarms), rather than a high recall value (i.e., predicting every single alarm).

5.5.1 Dataset

We first need to decide on an appropriate feature space. To predict the occurrence of an alarm at time t, we first of all make use of the values of each "signal" at this time. There are three kinds of signals: the states of the alarms (on/off), the states of control signals (on/off) and the read-outs of analogue sensors. When learning a decision tree

to predict the occurrence of alarm a, we can in principle make use of all three kinds of signals, including the states of other alarms $a' \neq a$. However, while alarms may be good predictors of other alarms, this information may not always be of interest to the process engineers: learning that, e.g., a "high high level" alarm on a tank can be predicted by a "high level" alarm will not interest them; on the other hand, learning that a failure in a specific component can be predicted by a failure in a different component may be of interest. Because a lot of domain knowledge is required in order to select alarms that may be of interest, we leave this choice to the engineer.

To predict the occurrence of an alarm at time t, we not only use the values of the available signals at time t, but also include some information about their history. This is needed because an alarm may be caused by a succession of events (e.g., a pressure alarm might be triggered if the temperature in a certain tank is higher than $70°C$ for more than 15 minutes). For each signal s, we introduce the following set of features about its history before t. For the boolean signals (i.e., states of alarms and control signals), we consider: (1) the elapsed time since the signal last went from off to on; (2) the elapsed time since the signal last went from on to off; (3) how many changes in its state occurred in the last 30 minutes. The first two of these features are capped at 60 minutes, because events longer in the past are deemed to be irrelevant by the process engineers. For analogue measurements, we consider the following six features: its current value; its average, minimum, maximum and standard deviation over the last 30 minutes; its increase or decrease over the last 30 minutes.

When predicting alarm a, we use each time point t at which a occurs as a positive example. For selecting negative examples, we need to consider a number of things. In principle, we could use all time points at which a does not occur as negative examples. However, since alarms occur relatively infrequently, the number of time points t' at which an alarm does not occur vastly outnumbers those at which it does. Including all such t' as negative examples would create a very large and unbalanced dataset. On the one hand, such extreme imbalance in training data has a serious negative impact on the performance of machine learning techniques. On the other hand, the immensity of the training set (i.e., multiple billions of training examples) makes it computational infeasible to provide the desired functionality in an acceptable amount of time. A solution for both of these problems is under-sampling the majority class (i.e., non-alarms). A possible downside of under-sampling a dataset is lowering the diversity of the training examples. This is something we should try to avoid.

The tank terminal's state is a slowly changing process. Consequently, samples of the terminal state that are taken on times close to each other are very similar and therefore provide only limited value for machine learning techniques. In order to select samples that are representative for the whole dataset, we should try to spread the samples as much as possible.

Likewise, the terminal state at times t' close to t is very similar to that at t. Moreover,

there may be some delay involved in the generation of the alarms (e.g., an alarm might only be raised if a sensor read-out exceeds a threshold for a certain time period). Therefore, actual cause of the alarm might already be present at certain time points $t' < t$, even though the alarm itself has not yet been generated at t'. The terminal state at such t' close to t Including such time points t' as negative examples would introduce a risk of severely confusing the learning algorithm.

In order to find examples as diverse as possible, without including samples that too closely resemble alarm states, we generate examples as follows: we sample the state of the terminal once every hour, if alarm a occurs within one hour of our chosen sample time, we discard the sample; otherwise, we include it as a negative example. Since we are learning from data that was collected over a period of one year, this method still produces a copious number of negative examples. For example, a typical alarm produces a dataset with 80 positive examples versus 4000 negative examples, each having 4000 features. The fact that this dataset is unbalanced may be a challenge for the learning algorithm; we return to this issue later in this chapter.

5.5.2 Algorithms

To learn the decision trees, we have used algorithms from Spark [48], Scikit-learn [54] and Weka [31]. Here, the following points are of importance.

Interpretability

Our main goal is to produce decision trees that help the process engineer analyse alarms on the terminal, by alerting him to patterns that he might not have noticed otherwise. Over the course of this project, we have closely collaborated with the engineers in order to discover which kind of decision trees are best suited for this purpose. A first observation is that the depth of the tree plays a crucial role. Depths larger than three turned out to be very difficult for the process engineer to interpret. A second observation is that the engineers are mainly interested in nodes with a large positive bias (i.e., nodes with a high ratio of alarms). Combining these two observations, the most interpretable trees can be achieved by minimizing the depth of positive leafs.

Splitting Criterion

Decision tree learning algorithms typically function by repeatedly selecting one of the current leaves of the tree and splitting this leaf into two child nodes. An important question is how—i.e., based on which feature—the leaf should be split. The intuition is to do this such a way that the most distinctive split between positive and negative

examples is achieved. This quality of a split can be measured in different ways, e.g., by gini impurity or entropy gain. While these two criteria produce different results when learning multiple classes [12], the difference between them is negligible in settings—like ours—with only two classes [56]. Therefore, they produce trees in which the depth of positive nodes is comparable, thus of equal legibility. Because it is easier to calculate, we use gini impurity.

The impurity of a node N which covers N_+ positive examples and N_- negative ones is defined as $1 - \left(\frac{N_+}{N_+ + N_-}\right)^2 + \left(\frac{N_-}{N_+ + N_-}\right)^2$, which simplifies to the formula:

$$Gini_N = \frac{2N_+N_-}{(N_+ + N_-)^2}$$

A decision tree is then constructed by splitting the dataset into two child nodes C_1 and C_2, based on a splitting criterion. This criterion is selected from all possible features, based on how well it is able to produce pure children. The metric for this is a weighted sum of the gini impurities of the resulting child nodes WI_C (the weighted impurity of a certain criterion) , i.e.,

$$WI_C = \frac{C_{1+} + C_{1-}}{P} Gini_{C_1} + \frac{C_{2+} + C_{2-}}{P} Gini_{C_2}$$

Where $P = C_{1+} + C_{1-} + C_{2+} + C_{2-}$ is the number of examples in the parent node. When filling in the formula for the *Gini* impurity:

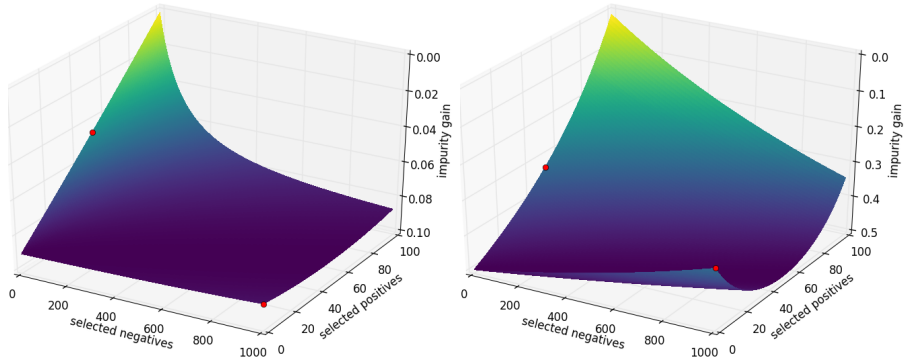
$$WI_C = \frac{2}{P} \times \left(\frac{C_{1+}C_{1-}}{C_{1+} + C_{1-}} + \frac{C_{2+}C_{2-}}{C_{2+} + C_{2-}} \right)$$

From all possible splitting criteria, the one with the lowest WI_C is then selected.

Balancing Data

As mentioned before, alarms are rare events, which leads to highly unbalanced datasets, with many more negative examples than positive ones. A common method to cope with this in decision tree learning is to reweigh the data set: if there are pos positive and neg negative examples, each positive example is assigned a weight of $\frac{total}{2 \times pos}$, while each negative example is assigned a weight of $\frac{total}{2 \times neg}$, where $total = neg + pos$. In this way, the total weight of both classes is equal. The weighted impurity of a criterion in a tree trained on such a reweighed dataset is:

$$WI_C = \frac{total}{P} \times \left(\frac{C_{1+}C_{1-}}{C_{1+} \times neg + C_{1-} \times pos} + \frac{C_{2+}C_{2-}}{C_{2+} \times neg + C_{2-} \times pos} \right)$$



(a) Unbalanced data.

(b) Balanced data.

Figure 5.5: Comparison of the weighted impurity in decision trees trained on unbalanced and balanced data.

When a dataset is highly skewed, typically, higher accuracy can be achieved by balancing the dataset [68]. However, as a downside in our setting, the depth of positive nodes tends to increase when a decision tree is learned from balanced data, making the tree less readable. We illustrate this with an example.

Suppose an alarm has occurred 100 times and that we have generated 2000 negative examples. Each possible way of splitting a root node into two children can be characterized by the pair (p, n), where p and n are respectively the number of positive and negative examples covered by the right child of the root (the left child will then contain $100 - p$ positive and $2000 - n$ negative examples). Figure 5.5 plots the weighted impurity for each of these possible pairs (p, n), for a tree trained on balanced and unbalanced data. Because these graphs are obviously symmetrical, only the half for which $n \leq 1000$ is shown.

Overall, the weighted impurity is higher in the tree learned from balanced data, but this is simply because the reweighing raises the impurity of the dataset. A more interesting observation is the following. In a tree trained on unbalanced data, separating nodes with a high positive bias (i.e., nodes that contain a large number of alarms) results in a lower weighted impurity than separating nodes with a negative bias. For instance, splitting a node with a distribution $(100, 2000)$ into $(50, 0)$ and $(50, 2000)$ would result in a significant (small) weighted impurity, this split is indicated by the leftmost red dot in Figure 5.5a . A split into $(0, 1000)$ and $(100, 1000)$ results in an insignificant (large) weighted impurity, as indicated by the bottom right dot in Figure 5.5a. In other words, when the learning algorithm is faced with the choice between a splitting off a leaf $(50, 0)$ or splitting off a leaf $(0, 1000)$, it will favour the former. As a result,

learning on an unbalanced dataset favours splitting criteria that separate alarms in an early stage, and the depth of positive nodes will be small in such a tree.

In a tree learned from balanced data, by contrast, nodes with a negative bias can also yield small weighted impurity: indeed, as can be seen in Figure 5.5b, the reweighing causes the split into $(50, 0)$ and $(50, 2000)$ to score precisely as good as the split into $(0, 1000)$ and $(100, 1000)$. So, while the tree trained on unbalanced data has a preference for nodes with a positive bias, balancing the data removes this preference from the tree learning algorithm.

As mentioned in Section 5.5.2, the process engineers are mainly interesting in trees that have nodes with a large positive bias at a low depth. Consequently, unbalanced learning is best suited for our purposes. This is indeed also what we found when comparing both methods in practice, together with the process engineers.

5.5.3 Visualization

Tree Representation

When discussing the learned decision trees with the process engineers, it became apparent that they still found these hard to interpret. In addition, they were often distracted by less relevant parts of the trees. In an attempt to make the trees easier to interpret, we developed a visualization with the following features. The thickness of an edge from parent P to child C is proportional to the fraction of positive examples covered by P that is still covered by C. When reading the tree from top to bottom, this allows to easily keep track of in which branches most of the alarms occur. The thickness of a node's border is proportional to the importance of that feature throughout the complete tree. This makes it easy to see at a glance which features are most relevant for predicting the occurrence of the alarm, even when these features get used in different parts of the tree. Finally, nodes that cover more than a given number of positive examples are highlighted.

In addition to these visualizations, we also allow the user to hide parts of the tree, in order not to be distracted by less relevant parts. In particular, it is possible to: (1) hide nodes that cover fewer positive examples than a given threshold; and (2) hide subtrees of nodes whose ratio of covered positive to covered negative examples is outside a specified range. On the one hand, if the ratio exceeds the maximum, the node represents a set of circumstances in which the alarm is already far more likely than usual. Such states are worthy of investigation by the process engineer, regardless of what still might be present further down in the tree. On the other hand, if the ratio falls below the minimum value, than the node represents a set of circumstances in which it

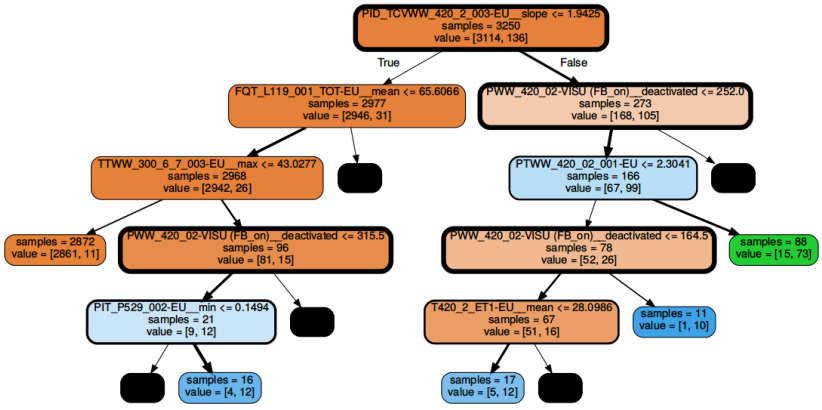


Figure 5.6: Visualization of a decision tree.

$Alarm_{ID}$ occurred in 73 out of 88 cases in which:
    the last half hour the value of $sensor_1$ increased
       by more than 1.9;
    $signal_1$ became inactive less than 252 minutes ago;
    the value of $sensor_2$ lies between 2.3 and 2.9.
This describes 53% of the 136 occurrences of this alarm.

is very unlikely that an alarm will occur, so further expansion of this node is unlikely to be interesting.

Rule Representation

Even with these visualizations, the engineers continued to find it hard to locate the information they were most interested in (i.e., in which circumstances does the alarm typically occur) and to interpret the information in the tree. Therefore, we also developed an alternative representation, in which a branch of the tree is representation as a natural language rule, in the following format:

Such a rule is constructed by iterating over all decisions in the branch. Each time a new feature is encountered, a new clause is added to the rule. When a feature reappears, the corresponding clause is updated to take the new decision into account. We convert all branches to rules and list them in descending order of number of covered alarms, so that the most interesting rules appear first.

Correlated Features

A problem with the use of decision trees arises when multiple highly correlated features contribute to an alarm: a decision tree will typically contain only one of these, while the process engineer would like to see them all. For this reason, we also allow the user to train an entire *random forest* of multiple decision trees. In the presence of highly correlated features, the randomness of the forest will make different trees use a different feature from the correlated set. We then generate rules for the most interesting branches in this entire random forest.

5.5.4 Graphical Interface

Figure 5.7 shows a screenshot of our prototype. It allows the user to:

1. Select which subsystem (e.g., the acetate system) to investigate.
2. Save and later reload the current configuration.
3. Select the alarm for which he wishes to learn a decision tree.
4. Force the system to rebalance the dataset (as motivated above, this is off by default) and choose a specific decision tree learning algorithm (the default is that of Scikit-learn, which appears to produce the most readable trees).
5. Choose tree or rule representation of either a single tree or random forest.
6. Exclude features from the tree. This is useful because an alarm typically has a number of trivial predictors. For instance, a “high temperature” alarm can be explained perfectly by the value of a particular temperature sensor. Including such a feature will produce a single-node decision tree with perfect accuracy, which is obviously of little value to the process engineer. Because of the absence of a model of the terminal, the system cannot automatically remove this kind of features and this task must be left to the user.
7. Prune certain parts of the tree, as already explained above.

5.6 Results

Our system has been tested by the process engineers. For the first component of the system, the engineers found that the statistics and visualizations indeed enabled them

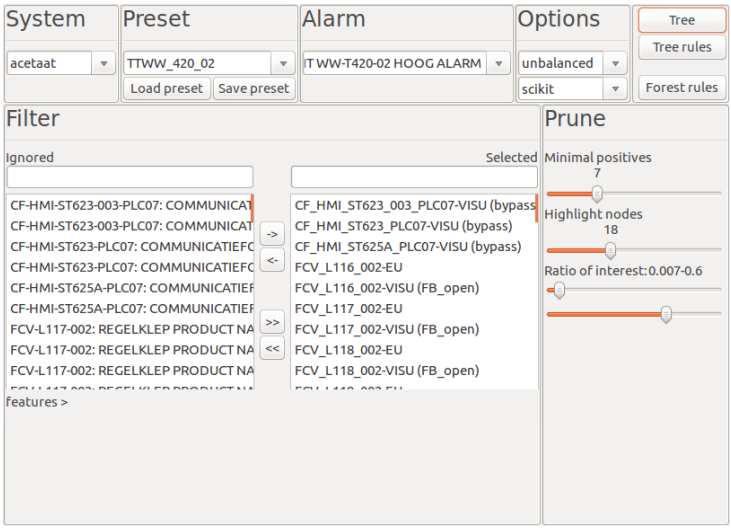


Figure 5.7: GUI for the decision tree learner.

to identify interesting “lingering” alarms. They therefore judged this component to provide a relevant added value.

The second component was more challenging. For our initial prototypes, the engineers found it difficult to interpret the decision trees. After the improvements that were outlined in the previous two sections, the engineers eventually had a more positive reaction. In the final prototype that we delivered, the engineers found it considerably easier to identify relevant information. Moreover, they also agreed that the information in the trees was typically indeed indicative of the underlying causes for the alarms. However, they also found that most of this information was typically either already evident to them from their own experience, or that it would have been easily obtained by manually inspecting the logs. They therefore judged that the system did not provide enough added value to warrant its being integrated into their production environment.

5.7 Conclusions And Future Work

This chapter reports on a research project in collaboration with the company *Agidens* and an oil tanking terminal in the port of Antwerp, Belgium. Our goal was to investigate the use of Artificial Intelligence technology to reduce the workload of control room operators. Several characteristics of this terminal make it impossible or undesirable to

$Alarm_{ID}$					
C+	**Increase of** $sensor_1$ **the last half hour**	$signal_1$ **became inactive**	$sensor_2$	$Alarm_{ID}$	**Not** $Alarm_{ID}$
1	> 1.9	< 252 minutes	$[2.3, 2.9]$	73	15

Figure 5.8: Table representation of an alarm prediction.

apply standard methods, such as decision support by means of root cause analysis, or rule-based alarm reduction.

Therefore, we focused on the process engineers, in order to help them reduce the number of alarms that occur. We developed a system consisting of two components: the first visualizes the alarms in order to allow the process engineers to identify lingering alarms that may be beneficial to investigate; the second applies decision tree/forest learning methods in order to detect patterns that might help the process engineers figure out the general causes for certain kinds of alarms, such that the conditions in which these alarms occur might be avoided in the future. For this purpose, interpretability of the learned models is crucial. We therefore developed different visualizations of the models.

When evaluating the prototype of the system, the engineers saw a clear benefit in the first component. The second component was found to be usable and to produce relevant results, but provided only limited added value.

We conclude that it is technically possible to build an interpretable model using knowledge that can be found in raw data. In this use-case, however, the constructed model did not present sufficient results to teach domain experts additional domain knowledge. This is due to two factors. First, because domain experts already have a thorough knowledge of the domain, discovering knowledge that can supplement their expertise is difficult. Second, because of the vast number of logged parameters, lack of a terminal model, and the amount of noise in the data (e.g. multiple concurrent operations), the data in this use-case is suboptimal.

An interesting alternative to representing the learned patterns in natural language rules, would be to represent this knowledge in a DMN-based format, such as the table in Figure 5.8. However, because the collaboration with the tank terminal company has ended, we were not able to validate this. Therefore we did not yet investigate this further, but it would be an interesting topic for future research nevertheless.

Moreover, future work could examine the influence of other splitting criteria on the performance and interpretability of decision trees. Additionally, further analysis of the trade-off between performance and interpretability could be performed.

Chapter 6

Conclusions and Future Work

6.1 General Conclusions and Lessons Learned

This dissertation demonstrates the practical use of Knowledge Representation technologies to assist domain experts. By drawing on case studies displaying a broad variety, both in application domain and user expectations, we show that the proposed techniques can provide valuable contributions to the experts' wide-ranging expectations.

The research is motivated by the many shortcomings we identified in the existing approaches, whilst conducting the case studies described above. First, existing systems typically focus on replicating the tasks of human experts, while little attention is paid to supporting them with those tasks. Second, even though a readable representation of knowledge facilitates knowledge extraction and improves transparency and maintainability of such systems, readability is rarely considered in such systems. Systems that do focus on readability, on the other hand, typically lack the required functionalities. Third, a body of research focuses on how to support multiple functionalities on the same knowledge. However, these techniques too rarely find their way to practically usable systems. Moreover, knowledge in such systems is typically specified in complicated formalisms, which makes them illegible for domain experts that have limited experience with such languages.

Throughout this dissertation, we examine how techniques from the field of knowledge representation can be used to cope with these shortcomings. The Knowledge Base Paradigm states that the knowledge in a system can be separated from the functionalities that are offered. In other words, the domain knowledge can be modelled in an expressive logical language and the resulting model can then be used by a number of inference

tasks. This approach allows experts to use a number of customized functionalities that offer them a wide-ranging support, all based on the domain knowledge specification. By providing an expressive, intuitive and readable notation for such a logical language, we can provide this support based on a knowledge specification that is constructed in a readable way. Moreover, we show that historical data can be used in addition to or instead of expert know-how to provide additional support for domain experts.

We can conclude that KR can indeed make useful additions that facilitate the daily jobs of engineers. In the case study conducted in Chapter 3, we carried out a thorough evaluation of the practical utility of an expert support system. This evaluation yielded numerous advantages of our proposed system. First, during the knowledge elicitation step, it forces the experts to reflect on the reasons why they make certain choices. This increases their understanding of the problem domain. Second, the creation of such a knowledge base improves standardization of proposed solutions. I.e., it helps to avoid situations where different engineers propose different solutions for the same problem. Third, such a system is useful to transfer knowledge from more experienced engineers to less experienced ones. Particularly because of the "report feedback" functionality, which allows the users to continuously update the knowledge in the tool. The engineers identified the system as being particularly useful for less experienced engineers as a learning tool. Fourth, because they are supported in a multitude of ways – such as providing them with insight in the reasoning of the system and allowing them to explore historical solutions – engineers predict this tool will help them to continuously learn about the domain. Fifth, engineers indicate that using such a system would encourage them to be more critical about their decisions, which increases their confidence in the selected solutions. Finally, it allows engineers to find solutions faster, and it also increases the number of first-time-right solutions. This not only saves them a tremendous amount of time, it also helps to increase customer confidence, which is even more important.

We look back on each of the research topics outlined in Figure 6.1, and use this figure to structure our conclusions.

6.1.1 Sources of knowledge

We consider two possible sources of knowledge: expert know-how and data.

Collecting these different kinds of knowledge requires effort in different areas. *Expert know-how* can be elicited during knowledge elicitation workshops. This method is labour-intensive for the experts, but it allows to focus on the most important aspects of the knowledge. Keeping this knowledge up to date remains a manual process. To collect *data*, an initial effort is needed to set up a logging database. Moreover, a

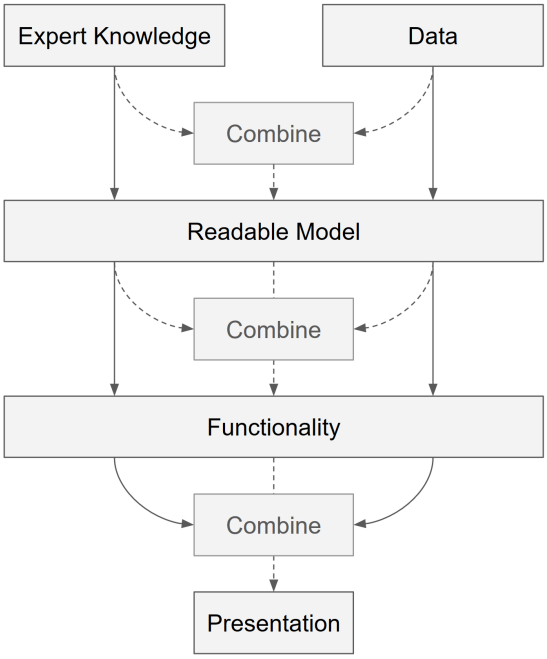


Figure 6.1: Graphical representation of the research topics in this dissertation.

procedure should be thought of to keep this database up to date. When this setup is established, minimal effort of the expert is needed to keep this database up-to-date.

Using knowledge of these different sources implies a completely different approach. During the extraction process of expert know-how, it is possible to focus on expressing the knowledge in such a way that allows reasoning with this knowledge in multiple ways. Because experts are involved in the development of such models, an additional advantage is that there is more control over the correctness of the constructed models. Moreover, because experts are actively involved in the creation of such a system, it increases the experts' understanding, not only of the system, but also of the domain knowledge itself.

On the other hand, when building a *data-based* system, a multitude of data is gathered. This data is then used for data exploration, statistical analysis or to create a predictive model (such as decision trees). This approach does not require involvement of the experts during the development of the knowledge models. However, because of this, there is no guarantee that the learned knowledge is useful or correct.

We conclude that using expert know-how simplifies obtaining requested functionalities

and readable models. Even though the extraction process is more laborious, it is worthwhile to utilize this knowledge if it is available. The use of data to complement this expert know-how can definitely provide an effective solution if the latter is not sufficient, especially if data has already been gathered. The predictive power of Machine Learning techniques is undeniable, they show promising results in a multitude of domains. However, for our purpose, i.e., supporting domain experts with readable models and broadly usable functionalities, they provide only limited utility.

6.1.2 Readability

Readability of Expert Know-How Models Throughout this dissertation we focused on representing expert know-how in a readable way. We identified three essential advantages. First, during knowledge elicitation, it is crucial to have mutual understanding of the knowledge that is being discussed. A readable representation of the knowledge facilitates this elicitation step greatly. When using a notation that is directly executable, an additional advantage at the elicitation stage is that the constructed model can directly be used in the implementation, i.e., no manual translation to another representation should be made. Second, keeping knowledge in such a model up-to-date is easier because experts are able to interpret the knowledge in it and they were closely involved in creating this model. Third, because experts can easily verify correctness of the represented knowledge, it improves their confidence.

There are multiple options to express decision logic in a user-friendly notation. In Chapter 3 we examined whether the DMN standard can provide a solution for this. We found that this notation succeeds in representing decisions in a structured, readable way. When the experts have a step-by-step procedure to create a solution, this notation is very useful. However, a procedure is not always readily available. The use of DMN forces to invent such a procedure, which is difficult on occasions. Sometimes it is (almost) impossible to formulate, i.e., the procedure would introduce so much complexity that the model would become unreadable. Moreover the knowledge in a DMN model can only be used in one way, as we describe in Section 6.1.3 When no procedure has been established to solve a certain problem, or when multiple functionalities are required, we discourage the use of DMN.

In Chapter 2, we proposed the cDMN notation, which is based on the DMN standard. This notation displays similar readability as regular DMN, while it allows to express more complex types of knowledge. Moreover, it does not force the user to invent a procedure to obtain a solution. Furthermore, the knowledge in a cDMN model can be formulated in such a way that multiple functionalities can be executed on it. We analysed the practicality of this notation by modelling a variety of problems from multiple domains, including the knowledge of the case studies in Chapter 3 and 4. The

cDMN notation proves to be very expressive, while on the other hand, it allows to represent knowledge in a compact and readable way.

We conclude that the cDMN notation allows to represent knowledge in an expressive way, which enables to use multiple functionalities on the represented knowledge. Moreover, the notation allows to represent knowledge without forcing to find a procedure to come to conclusions, and foremost, the notation is user-friendly to the extent that it can be used by domain experts.

Readability of Data-based Models In Chapter 5 we briefly touched upon the topic of readable models based on data. We showed that it is indeed possible to produce readable models from data alone. However, reasoning with such models is limited. They merely serve as a way to present the knowledge found in the data.

We conclude this section with stating that expert know-how provides a better basis to produce readable models that are able to provide more functionalities.

6.1.3 Functionality

Which functionalities should be supported greatly depends on the requirements of the user in a specific domain. Consequently, the case studies examined in this dissertation also have different functionality requirements.

In Chapter 3, we conducted an extensive study of which functionalities are useful to support domain experts in their daily jobs. The nature of the represented knowledge determines what functionalities can be supported. When knowledge is represented as a *deterministic procedure*, only a limited number of functionalities can be supported, e.g., transform a complete set of inputs into an output. In principle, the same knowledge can be used to determine which inputs could have led to a specified output. However, this kind of functionality should be used with caution, i.e., instead of answering the question "in which circumstances could this output possibly be used?", it answers the question "in which circumstances will this output be selected by the deterministic procedure?", which is an important nuance. A system based on this kind of knowledge manages to effectively reduce the expert's load by automating parts of his job. However, such a system is of little use to provide more extensive assistance, for a number of reasons. First, meaningful conclusions can only be drawn when a complete set of inputs is provided, in real-life settings this is often too limiting. Second, such a deterministic system produces only one solution for a given set of inputs. It provides no information about alternative solutions. Third, technically a different procedure could be determined to support another functionality, however, this requires the development of a completely new model for each needed functionality. These factors make such a system useless as a decision support system.

Knowledge that is formulated in a constraint-based model, lends itself more to this purpose. By making use of the Knowledge Base Paradigm, the knowledge in such a model can be utilized in a number of different ways, such as

- reasoning with partial inputs, or even from output to input,
- comparing consequences of choices,
- explaining why certain deductions are made,
- changing the underlying model while using it, e.g., by ignoring constraints.

Moreover, because the model can reason in such different ways, it allows to align the results of ML algorithms in combination with the knowledge base, which enables to use functionalities based on historical data, such as data exploration and active learning.

We conclude that expressing knowledge using a constraint-based model has unmistakable advantages in the area of functionality. Moreover, we demonstrate the practical applicability of the knowledge base paradigm in real-life industrial applications.

In Chapter 4, this comprises mainly the automation of the montage process. In principle the same knowledge could be used to determine the quality of montages, or to learn parameters of the model. However, we concluded that this was computationally infeasible.

Chapter 5 describes a data analysis system that is designed to help domain experts to identify which alarms are worth to be investigated, and on the other hand enable them to inspect these interesting alarms and analyse their cause.

6.1.4 Presentation

In addition to the readability of represented model knowledge, we observed that the presentation of results is crucial. Regardless of how many useful functionalities a system provides, they are useless for a user if not presented in a user-friendly way, as we found in Chapter 3 and Chapter 5. The presentation is two-fold. A first concern is to allow the user to specify the relevant input data and activate the requested functionalities in a convenient way. A second concern is to present the results of the system in an understandable way to the user. How the results are presented and the system interaction is defined, depends greatly on the functionalities the system has to offer. In any circumstances, the use of a GUI considerably improves user experience.

6.1.5 Combining Knowledge Sources

Generally, experts have thorough knowledge about their domain. Accordingly, it is a logical choice to utilize this knowledge. However, even for domain experts it is impossible to have complete knowledge. Complementing the knowledge of experts with information found in data typically contributes to a more complete system.

Multiple ways to combine knowledge from those different sources can be distinguished. Chapter 3 describes the creation of two separate systems for each of these knowledge sources, each with their own functionalities. These independent functionalities are then aligned to cooperate and present them in to the user in one combined system. This is a promising approach, and the integration of these components ensures a pleasant user experience. Even though functionalities based on both knowledge sources are available in one system, there is a clear distinction between functionalities based on expert know-how and data. This ensures that the user can select which source he considers most useful for his current task.

In Chapter 4, we attempted to combine knowledge of the two different sources into one model. Even though, in theory this appeared to be a promising option, we have to conclude that in practice this approach is not feasible. The expert know-how based model turned out to be too complex to fine-tune it with data.

In the future work in Section 6.2, a number of additional possibilities to combine are discussed to combine knowledge form those different sources.

6.2 Future Work

During the course of this PhD, we proved that knowledge representation can be used to tackle a number of problems in a broad variety of case studies. However, there is still a large body of future work that could be researched to improve the solutions proposed in those case studies. We structure this future work by discussing each of the chapters individually.

cDMN

We propose a novel notation based on the DMN standard called cDMN. This notation elegantly represents constraint-based knowledge. In the conclusion of Chapter 2, we identified a number of shortcomings of this notation.

First, cDMN's expressivity is restricted, i.e., a number of logical constructs, such as existential quantification and disjunctions in the output of constraint tables, currently

cannot be compactly represented. It would be useful to investigate how such additional expressions can be represented in a compact and readable manner, and which of these constructs are necessary to cover more applications.

In this dissertation, we tested the usability of the proposed notation in various use cases. However, it would be useful to extend the tested cases to analyse the applicability of the cDMN notation in a multitude of different domains.

Another interesting topic is to have a tight integration of the cDMN notation and the AutoConfig tool presented in Chapter 3. This would create a complete pipeline that enables experts (1) to model domain knowledge in a readable notation and (2) use the AutoConfig tool to reason with this expressed knowledge in a user-friendly way.

Machine Component Design

In Chapter 3 we propose two systems to cope with the different requirements of the sales department and the engineering department. The first system is used by the sales department to deterministically find a solution, based on a set of customer requirements. The second system offers a more extensive decision support system to assist engineers to design machine components.

Keeping knowledge in these two systems aligned is important. In our current proposal this remains a manual task. It would be interesting to determine a specific automated procedure for this. Because the knowledge in the constraint-based system is more general, our current view on this topic is that a knowledge consolidation step is needed to extract a deterministic procedure from this constraint based knowledge. Of course readability of the knowledge in the resulting model is key. Moreover, such a knowledge consolidation procedure could be used to find a deterministic process for other application areas as well.

An additional interesting topic is how to use user feedback to automatically propose adaptations to the knowledge base. This would facilitate the task of the knowledge base maintainer.

The decision support system analyses historical data and presents it to the user. An additional use of this data could be to identify constraints that could be used to supplement the knowledge in the constraint-based knowledge base.

We currently have implemented an ad-hoc method to cope with missing data. Future research could be conducted on finding a standardized approach for this.

Moreover, the current knowledge base constructed in this project is based on the IDP system. However, this system does not handle floating-point calculations very well. In

future work, it would be interesting to use an inference system that is able to reason with floating-point numbers.

The AutoConfig tool offers a multitude of functionalities on the constructed knowledge base. However, further research could be conducted on how these functionalities could be presented to the user in a more intuitive way.

Last, the constructed knowledge base includes a number of application areas. By extending the covered application areas, more common constraints will surface, which allows for a more generally applicable knowledge base.

Automatic Video Montage

Chapter 4 describes a probabilistic model that makes automatic video montages. As already discussed in the conclusion of this chapter, there are a few topics that are interesting to be further investigated.

First, we experimented with fine-tuning the probabilistic parameters used in the montage model, making use of data. This would have allowed us to train different models for other scene settings. However, because of computational complexity, this calibration did not succeed. In future research, it could be interesting to look into other algorithms to fine-tune these parameters.

We provided a proof of concept that makes it possible to represent probabilistic knowledge in the cDMN notation. It would be very useful to further investigate this path in future research.

Alarm Management

Chapter 5 discusses an algorithm to identify interesting alarms, as well as a method to analyse such alarms, and present those results in a readable way to the user.

One thing we did not yet investigate is how to represent decision logic learned from data in a DMN-like format and how such conversion could improve the interpretability of the discovered logic. Moreover, future research could further explore the trade-off between performance and interpretability of decision trees. An additional interesting topic of future research is the effect of different splitting criteria on the interpretability and performance of decision trees.

Bibliography

[1] A. Abraham. “Rule-Based expert systems”. In: *Handbook of measuring system design* (2005).

[2] B. Abu-Nasser. “Medical expert systems survey”. In: *International Journal of Engineering and Information Systems (IJEAIS)* 1.7 (2017), pp. 218–224.

[3] B. Aerts, M. Deryck, and J. Vennekens. “A Comparing Study of Deterministic, Constraint-based and Hybrid methods in the Context of Machine Component Design (submitted)”. In: *Expert Systems with Applications* (2020).

[4] B. Aerts, T. Goedemé, and J. Vennekens. “A probabilistic logic programming approach to automatic video montage”. In: *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. IOS Press. 2016, pp. 234–242.

[5] B. Aerts, T. Goedemé, and J. Vennekens. “A Probabilistic Logic Programming approach to Automatic Video Montage (Extended Abstract)”. eng. In: The Eurographics Association; Goslar, 2017. ISBN: 978-3-03868-031-4.

[6] B. Aerts, K. Van Dessel, and J. Vennekens. “Alarm Management on a Liquid Bulk Terminal”. In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer. 2017, pp. 279–285.

[7] B. Aerts, S. Vandevelde, and J. Vennekens. “Tackling the DMN Challenges with cDMN: a Tight Integration of DMN and Constraint Reasoning (submitted to RuleML+RR2020)”. In: *arXiv preprint arXiv:2005.09998* (2020).

[8] B. Aerts and J. Vennekens. “Application of Logic-Based Methods to Machine Component Design”. In: *Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018)*. 2018.

[9] J. Ahnlund, T. Bergquist, and L. Spaanenburg. “Rule-based reduction of alarm signals in industrial control”. In: *Journal of Intelligent and Fuzzy Systems* 14.2 (2003), pp. 73–84. ISSN: 1064-1246.

[10] M. Alviano, F. Calimeri, G. Charwat, M. Dao-Tran, C. Dodaro, G. Ianni, T. Krennwallner, M. Kronegger, J. Oetsch, A. Pfandler, et al. "The fourth answer set programming competition: Preliminary report". In: *Logic Programming and Nonmonotonic Reasoning*. Springer, 2013, pp. 42–53.

[11] B. Bogaerts, M. Bruynooghe, G. Janssens, and M. Denecker. "Predicate Logic as a Modelling Language: The IDP System". eng. In: *arXiv.org* (2018). URL: http://search.proquest.com/docview/2071739773/.

[12] L. Breiman. "Technical Note: Some Properties of Splitting Criteria". In: *Machine Learning* 24.1 (1996), pp. 41–47. ISSN: 1573-0565. DOI: 10.1023/A:1018094028462.

[13] D. C. Brown. "Defining configuring". In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 12.4 (1998), pp. 301–305. ISSN: 0890-0604. DOI: 10.1017/S0890060498124034.

[14] M. Bruynooghe, H. Blockeel, B. Bogaerts, B. De Cat, S. De Pooter, J. Jansen, A. Labarre, J. Ramon, M. Denecker, and S. Verwer. "Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with IDP3". In: *Theory and Practice of Logic Programming* 15.6 (2015), pp. 783–817. DOI: 10.1017/S147106841400009X.

[15] D. Calvanese, M. Montali, M. Dumas, and F. Maggi. "Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge". English. In: *Theory and Practice of Logic Programming* 19.4 (2019), pp. 536–573. ISSN: 14710684.

[16] P. Chakravarty, S. Mirzaei, T. Tuytelaars, et al. "Who's Speaking?: Audio-Supervised Classification of Active Speakers in Video". In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. ACM. 2015, pp. 87–90.

[17] P. Chakravarty and T. Tuytelaars. "Cross-modal Supervision for Learning Active Speaker Detection in Video". In: *arXiv preprint arXiv:1603.08907* (2016).

[18] D. B. Christianson, S. E. Anderson, L. He, D. H. Salesin, D. S. Weld, and M. F. Cohen. "Declarative camera control for automatic cinematography". In: *AAAI/IAAI, Vol. 1*. 1996, pp. 148–155.

[19] F. Daniyal and A. Cavallaro. "Multi-camera scheduling for video production". In: *Visual Media Production (CVMP), 2011 Conference for*. IEEE. 2011, pp. 11–20.

[20] I. Dasseville, L. Janssens, G. Janssens, J. Vanthienen, and M. Denecker. "Combining DMN and the knowledge base paradigm for flexible decision enactment". In: *RuleML 2016 Supplementary Proceedings*. 2016.

[21] B. De Cat, B. Bogaerts, M. Bruynooghe, G. Janssens, and M. Denecker. "Predicate logic as a modeling language: the IDP system". In: *Declarative Logic Programming: Theory, Systems, and Applications*. 2018, pp. 279–323. DOI: doi:10.1145/3191315-3191321.

[22] L. De Raedt, A. Kimmig, and H. Toivonen. “ProbLog: A probabilistic Prolog and its application in link discovery”. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 2462–2467.

[23] F. De Smedt, D. Hulens, and T. Goedemé. “On-Board Real-Time Tracking of Pedestrians on a UAV”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. Embedded Vision Workshop*. 2015.

[24] N. Decleyre, B. Aerts, and M. Deryck. *Combining DMN, First Order Logic and Machine Learning: The creation of Saint-Gobain Seals' Digital engineer (Extended Abstract)*. 2019. URL: https://dmcommunity.org/2019/10/05/notes-from-decisioncamp-2019/.

[25] M. Denecker and J. Vennekens. “Building a knowledge base system for an integration of logic programming and classical logic”. In: *Lecture Notes in Computer Science, International Conference on Logic Programming, ICLP, Udine, Italy, 9-13 December 2008*. Ed. by M. Garcia de la Banda and E. Pontelli. Springer, 2008, pp. 71–76. DOI: doi:10.1007/978-3-540-89982-2_12. URL: https://lirias.kuleuven.be/handle/123456789/235155.

[26] M. Deryck, B. Aerts, and J. Vennekens. “Adding Constraint Tables to the DMN Standard: Preliminary Results”. In: *International Joint Conference on Rules and Reasoning*. Springer. 2019, pp. 171–179.

[27] A. Diba, A. M. Pazandeh, H. Pirsiavash, and L. Van Gool. “DeepCAMP: Deep Convolutional Action & Attribute Mid-Level Patterns”. In: *CVPR 2016, International Conference on Computer Vision and Pattern Recognition*. 2016.

[28] L. Dubois, J.-M. Forět, P. Mack, and L. Ryckaert. “Advanced logic for alarm and event processing: Methods to reduce cognitive load for control room operators”. In: *IFAC Proceedings Volumes* 43.13 (2010), pp. 158–163. ISSN: 1474-6670.

[29] J. Feldman. “Representing and Solving Rule-Based Decision Models with Constraint Solvers”. In: vol. 7018. Jan. 2011, pp. 208–221. DOI: doi:10.1007/978-3-642-24908-2_23.

[30] C. Forza and F. Salvador. “Product configuration and inter-firm co-ordination: An innovative solution from a small manufacturing enterprise”. In: *Computers in Industry* 49.1 (2002), pp. 37–46. ISSN: 01663615. DOI: 10.1016/S0166-3615(02)00057-X.

[31] E. Frank, M. A. Hall, and I. H. Witten. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. http://www.cs.waikato.ac.nz/ml/weka/spark.apache.org/mllib/. 2016.

[32] Y. Fu, Y. Guo, Y. Zhu, F. Liu, C. Song, and Z.-H. Zhou. “Multi-view video summarization”. In: *Multimedia, IEEE Transactions on* 12.7 (2010), pp. 717–729.

[33] V. Gandhi, R. Ronfard, and M. Gleicher. “Multi-clip video editing from a single viewpoint”. In: *Proceedings of the 11th European Conference on Visual Media Production*. ACM. 2014, p. 9.

[34] A. Girgensohn, J. Boreczky, P. Chiu, J. Doherty, J. Foote, G. Golovchinsky, S. Uchihashi, and L. Wilcox. “A semi-automatic approach to home video editing”. In: *Proceedings of the 13th annual ACM symposium on User interface software and technology*. ACM. 2000, pp. 81–89.

[35] C. Gogos, P. Alefragis, and E. Housos. “Sensor enabled rule based alarm system for the agricultural industry”. In: *12th IEEE conference on Emerging Technologies & Factory Automation*. 2007, pp. 912–915.

[36] C. Grosan and A. Abraham. “Rule-based expert systems”. In: *Intelligent Systems*. Springer, 2011, pp. 149–185.

[37] A. Haug, L. Hvam, and N. H. Mortensen. “The impact of product configurators on lead times in engineering-oriented companies”. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM* 25.2 (2011), pp. 197–206. ISSN: 08900604. DOI: 10.1017/S0890060410000636.

[38] L. He, M. F. Cohen, and D. H. Salesin. “The virtual cinematographer: a paradigm for automatic real-time camera control and directing”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM. 1996, pp. 217–224.

[39] D. Hulens, B. Aerts, P. Chakravarty, A. Diba, T. Goedemé, T. Roussel, J. Zegers, T. Tuytelaars, L. Van Eycken, L. Van Gool, et al. “The CAMETRON lecture recording system: High quality video recording and editing with minimal human supervision”. In: *International Conference on Multimedia Modeling*. Springer. 2018, pp. 518–530.

[40] D. Hulens, T. Goedemé, and T. Rumes. “Autonomous lecture recording with a PTZ camera while complying with cinematographic rules”. In: *Computer and Robot Vision (CRV), 2014 Canadian Conference on*. IEEE. 2014, pp. 371–377.

[41] L. Hvam, M. Malis, B. Hansen, and J. Riis. “Reengineering of the quotation process: application of knowledge based systems”. In: *Business Process Management Journal* 10.2 (2004), pp. 200–213. ISSN: 1463-7154. DOI: 10.1108/14637150410530262.

[42] J. Jansen. “Model revision inference for extensions of first order logic”. In: *arXiv preprint arXiv:1405.4206* (2014).

[43] H. Jiang, S. Fels, and J. J. Little. “Optimizing multiple object tracking and best view video synthesis”. In: *Multimedia, IEEE Transactions on* 10.6 (2008), pp. 997–1012.

[44] KU Leuven Knowledge Representation and Reasoning research group. *The IDP framework reference manual*. English. KU Leuven. Sept. 2015. 29 pp.

[45] D. Lambert. “Zero-inflated Poisson regression, with an application to defects in manufacturing”. In: *Technometrics* 34.1 (1992), pp. 1–14.

[46] Q. Liu, Y. Rui, A. Gupta, and J. J. Cadiz. “Automating camera management for lecture room environments”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2001, pp. 442–449.

[47] V. Marek and M. Truszczynski. “Stable models and an alternative logic programming paradigm”. In: *The Logic Programming Paradigm: a 25-Year Perspective*. Ed. by K. Apt, V. Marek, M. Truszczynski, and D. Warren. Berlin: Springer, 1999, pp. 375–398.

[48] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. “MLlib: Machine Learning in Apache Spark”. In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 1235–1241. ISSN: 1532-4435.

[49] D. Mitchell, E. Ternovska, F. Hach, and R. Mohebali. *Model expansion as a framework for modelling and solving search problems*. Tech. rep. Technical Report TR 2006-24, School of Computing Science, Simon Fraser University, 2006.

[50] N. Nethercote, P. Stuckey, R. Becket, S. Brand, G. Duck, and G. Tack. “MiniZinc: Towards a standard CP modelling language”. In: *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*. Vol. 4741. LNCS. Springer, 2007, pp. 529–543.

[51] I. Niemelä. “Logic programs with stable model semantics as a constraint programming paradigm”. In: *Annals of Mathematics and Artificial Intelligence* 25 (1999), pp. 241–273.

[52] Object Modelling Group. *Decision Model and Notation*. Version 1.3. Dec. 2019. URL: http://www.omg.org/spec/DMN/.

[53] OpenRules, Inc. *OpenRules*. Version 6.4.2. May 20, 2017. URL: http://openrules.com.

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python ”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[55] Progress. *Corticon*. Version 6.1. Nov. 18, 2019. URL: progress.com/corticon.

[56] L. E. Raileanu and K. Stoffel. “Theoretical Comparison between the Gini Index and Information Gain Criteria”. In: *Annals of Mathematics and Artificial Intelligence* 41.1 (2004), pp. 77–93. ISSN: 1573-7470. DOI: 10.1023/B:AMAI.0000018580.96245.c6.

[57] M. Rezaeian. “Estimation Entropy and Optimal Observability”. In: *PerCom/PerSeNS conference*. 2006.

[58] Y. Rui, A. Gupta, J. Grudin, and L. He. “Automating lecture capture and broadcast: technology and videography”. In: *Multimedia Systems* 10.1 (2004), pp. 3–15.

[59] T. Sato and Y. Kameya. “PRISM: A Language for Symbolic-Statistical Modeling”. In: *Proceedings of IJCAI*. 1997.

[60] J. Thambirajah, L. Benabbas, M. Bauer, and N. F. Thornhill. “Cause-and-effect analysis in chemical processes utilizing XML, plant connectivity and quantitative process history”. In: *Computers & Chemical Engineering* 33.2 (2009), pp. 503–512.

[61] A. Trentin, E. Perin, and C. Forza. “Product configurator impact on product quality”. In: *International Journal of Production Economics* 135.2 (2012), pp. 850–859. ISSN: 09255273. DOI: 10.1016/j.ijpe.2011.10.023.

[62] D. Van Daele, N. Decleyre, H. Dubois, and W. Meert. “An Automated Engineering Assistant: Learning Parsers for Technical Drawings”. In: *arXiv preprint arXiv:1909.08552* (2019).

[63] S. Vandevelde and B. Aerts. *cDMN documentation*. 2020. URL: https://cdmn.readthedocs.io/en/latest/.

[64] S. Vandevelde and B. Aerts. *cDMN solver*. 2020. URL: https://gitlab.com/EAVISE/cdmn/cdmn-solver.

[65] S. Vandevelde, B. Aerts, and J. Vennekens. *cDMN: combining DMN with Constraint Reasoning (Extended Abstract)*. eng. 2020. URL: https://decisioncamp2020.home.blog/program/#Vandevelde.

[66] J. Vennekens, M. Denecker, and M. Bruynooghe. “CP-logic: A language of causal probabilistic events and its relation to logic programming”. In: *Theory and Practice of Logic Programming* 9.3 (2009), pp. 245–308.

[67] K. Wang, J. Xu, and D. Zhu. “Online root-cause analysis of alarms in discrete Bayesian networks with known structures”. In: *Proceeding of the 11th World Congress on Intelligent Control and Automation*. 2014.

[68] G. M. Weiss and F. Provost. “The effect of class distribution on classifier learning: an empirical study”. In: *Technical Report ML-TR 43, Department of Computer Science, Rutgers University* (2001). DOI: doi:10.7282/t3-vpw-sf95.

[69] J. Wittocx, M. Mariën, and M. Denecker. “The idp system: a model expansion system for an extension of classical logic”. In: *Proceedings of the 2nd Workshop on Logic and Search*. ACCO; Leuven. 2008, pp. 153–165.

[70] M. Yu, H. Yashio, J. Kikukawa, and N. Joo. *Rule based intelligent alarm management system for digital surveillance system*. US Patent 7,352,279. 2008. URL: https://www.google.com/patents/US7352279.

[71] L. L. Zhang. “Product configuration: A review of the state-of-the-art and future research”. In: *International Journal of Production Research* 52.21 (2014), pp. 6381–6398. ISSN: 1366588X. DOI: 10.1080/00207543.2014.942012.

About the Author



Bram Aerts was born on the November 3th in Duffel, in the year 1992. He started his university education in the year 2010 at campus De Nayer of the KU Leuven university. Four years later, he obtained the degree Master of Engineering Technology in Electronics-ICT (option ICT). His master thesis handled the automatic calibration of traffic cameras. In the course of his education, his interests were triggered in modelling knowledge bases in FO($\cdot$). When he was approached by his professor Joost Vennekens, he decided to start his PhD on the topic "Advanced Applications of Knowledge Representation Technology" as a member of the EAVISE research group. In this PhD he mainly focused on proving the practicality of KR technologies in industrial applications, with occasional additions of other technologies where needed. During his PhD, he collaborated with a number of industrial partners.

Apart from his doctoral research, Bram is passionate about music. He enjoys making music and playing with his band, this resulted already in a number of records. Moreover, he recently he gained interest in mixing and mastering music, which he put to use in two of his band's records.

List of Publications

Conference Proceedings

[1] B. Aerts, T. Goedemé, and J. Vennekens. "A probabilistic logic programming approach to automatic video montage". In: *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. IOS Press. 2016, pp. 234–242.

[2] B. Aerts, T. Goedemé, and J. Vennekens. "A Probabilistic Logic Programming approach to Automatic Video Montage (Extended Abstract)". eng. In: The Eurographics Association; Goslar, 2017. ISBN: 978-3-03868-031-4.

[3] B. Aerts, K. Van Dessel, and J. Vennekens. "Alarm Management on a Liquid Bulk Terminal". In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer. 2017, pp. 279–285.

[4] B. Aerts, S. Vandevelde, and J. Vennekens. "Tackling the DMN Challenges with cDMN: a Tight Integration of DMN and Constraint Reasoning (submitted to RuleML+RR2020)". In: *arXiv preprint arXiv:2005.09998* (2020).

[5] B. Aerts and J. Vennekens. "Application of Logic-Based Methods to Machine Component Design". In: *Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018)*. 2018.

[6] P. Carbonnelle, B. Aerts, M. Deryck, J. Vennekens, and M. Denecker. "An Interactive Consultant". In: *CEUR Workshop Proceedings* 2491 (Mar. 2020).

[7] N. Decleyre, B. Aerts, and M. Deryck. *Combining DMN, First Order Logic and Machine Learning: The creation of Saint-Gobain Seals' Digital engineer (Extended Abstract)*. 2019. URL: https://dmcommunity.org/2019/10/05/notes-from-decisioncamp-2019/.

[8] M. Deryck, B. Aerts, and J. Vennekens. "Adding Constraint Tables to the DMN Standard: Preliminary Results". In: *International Joint Conference on Rules and Reasoning*. Springer. 2019, pp. 171–179.

[9] D. Hulens, B. Aerts, P. Chakravarty, A. Diba, T. Goedemé, T. Roussel, J. Zegers, T. Tuytelaars, L. Van Eycken, L. Van Gool, et al. “The CAMETRON lecture recording system: High quality video recording and editing with minimal human supervision”. In: *International Conference on Multimedia Modeling*. Springer. 2018, pp. 518–530.

[10] S. Vandevelde, B. Aerts, and J. Vennekens. *cDMN: combining DMN with Constraint Reasoning (Extended Abstract)*. eng. 2020. URL: https://decisioncamp2020.home.blog/program/#Vandevelde.

Journal Articles

[1] B. Aerts, M. Deryck, and J. Vennekens. “A Comparing Study of Deterministic, Constraint-based and Hybrid methods in the Context of Machine Component Design (submitted)”. In: *Expert Systems with Applications* (2020).



FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
EAVISE
Jan Pieter De Nayerlaan 5
B-2860 Sint-Katelijne-Waver
b.aerts@kuleuven.be
http://www.eavise.be