KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Technology

Knowledge Base Systems in Practice: Approaches, Application Areas and Limitations

Marjolein Deryck

Supervisor:
Prof. dr. J. Vennekens

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

January 2022

Knowledge Base Systems in Practice: Approaches, Application Areas and Limitations

Marjolein DERYCK

Examination committee:
Prof. dr. M. Vanierschot, chair
Prof. dr. J. Vennekens, supervisor
Prof. dr. ir. M. Denecker
Prof. dr. J. Vanthienen
Prof. dr. M. De Backer (The Master Labs)
Prof. dr. K. Gerling
Prof. dr. M. Montali
(Free University of Bozen-Bolzano-unibz)

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

January 2022

© 2022 KU Leuven – Faculty of Engineering Technology
Uitgegeven in eigen beheer, Marjolein Deryck, Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne Waver, B-3001 Leuven (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

When I started my PhD trajectory four years ago, I thought that writing the preface would be the easiest part of my dissertation. In fact, as I like to think well in advance, I even figured out what it would look like. In one or maximum two sentences I would thank everyone who helped me to get here today. Possibly I would throw in a well-chosen quote. But in the end, time lacks to search for the perfect quote. And more importantly, now that my dissertation is finished and it is time to write the preface, I really like to take this opportunity to express my gratitude more elaborately.

First and foremost, I like to thank my supervisor Joost Vennekens. Thank you for hiring me, supporting me, putting me on the right track and guiding me throughout this entire process. I really enjoyed our inspiring discussions that led to this result. You gave me the freedom to search my own way and develop my own approach, but if I got stuck I could always count on you. It has been a pleasure working with you, and I hope our paths will cross again.

My gratitude also goes out to the members of my examination committee for the feedback on my work, suggestions for interesting research directions and improvements. Thank you all for your shown interest in my research. This thesis got better thanks to your comments. Prof. Marc Denecker, Prof. Jan Vanthienen and Prof. Manu De Backer, members of my supervisory committee, thank you for your active participation these last four years. I have been lucky to be able to explore a part of your rich expertise during my doctoral trajectory. Not only the actual discussions we had helped to shape this work. At various points during writing I asked myself "What would Marc, or Jan, or Manu say?"

A large part of this research could be executed with the combined funding of Vlaio and a group of Flemish companies in the form of a tetra project on Decision Analytics. This project led to a continued collaboration in the form of a C3 project funded by KU Leuven, and a short project funded by the National Bank of Belgium. Finally, this research also received funding from the Flemish

Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme. Thank you for making this research possible.

This dissertation would not have been what it is, if I could not have tackled real life cases at numerous companies. I really enjoyed working with every company, and with people who were enthusiastic to see the opportunities of our technology. During the regular user group meetings we created a network of companies interested in knowledge base systems. I am happy to see that some of you continue to collaborate also after the end of my research projects. I am also excited to continue working as an employee of Intelli-Select on the improvement and implementation of the prototype in the financial sector. Thank you all for your support, interesting cases, discussions, feedback and leads for future work.

For the doctor scheduling case I thank Pieter Smet, who shared his knowledge on scheduling, provided paper leads and constructive feedback.

And of course, my colleagues, who I could count on every day. Thank you for the (technical and other) support, for implementing my requests, for the numerous brainstorming sessions, for the nice coffee, for keeping my chocolates, for the paper leads, for making C204 the fun office, baking pancakes, ..., and so much more.

My PhD trajectory has been challenging and very rewarding, and each of you contributed to it in its own way. Thank you all for being a part of my path and hopefully we will see each other again in the future!

Abstract

In the last few decades our society has moved to a knowledge society in which continuous innovation has become more important than ever. Innovation is not the responsibility of R&D departments alone anymore, but is also driven by the knowledge of the company's expert employees.

In this thesis we investigate how expert knowledge can be used to develop powerful applications, and how applications should be designed to leverage the expert knowledge to the fullest. Following Liebowitz we look at this expert knowledge at three levels: the domain knowledge level that contains the factual knowledge of the domain, the inference knowledge to know how to execute a specific job, and task knowledge that refers to the ability of decomposing a complex task in subtasks [90]. This distinction aligns well with the lens of the Knowledge Base Paradigm (KBP) that we use. This paradigm advances a strict separation between the factual domain knowledge of a problem, and the inference tasks that use this knowledge to find a solution for a problem. In this distinction, the basic inference tasks are an example of inference knowledge, while selecting the correct inference tasks to solve a specific problem is an example of task knowledge.

Gathering these different kinds of knowledge to build a knowledge base system, requires different knowledge acquisition activities. With the help of six case studies we create a partial decision support system classification and derive knowledge acquisition recommendations for each of them.

The first category of systems is legal applications. These applications support users in making correct decisions based on statutory law. Of course, correctness and explainability of results are important requirements of legal systems. Contrary to common belief, dealing with legislation is not mainly characterised by deductive reasoning, but mostly by default reasoning with possibly incomplete information. This impacts the development and setup of the application, which should display a high level of interactivity. Product configuration systems

are the second kind of applications. They are not limited to the design of tangible products, but can also be used to develop intangible artefacts, such as a course program or communication templates. The help of domain experts to develop a configurator is indispensable, and a configuration project should explicitly focus on taking domain experts in the loop. The third category of applications is scheduling applications. The scientific body of knowledge on personnel scheduling is vast, but in comparison its use in practice is limited. We find that not only the quality of the schedule is important, but also the social role that many planners occupy. The last category is an overarching category of third-party applications, in which the company that develops the application does not have access to all knowledge. A more or less significant part of the knowledge is only available for the third party –often an external client– that uses the application. Applications in this category should devote a great deal of attention to making the knowledge base accessible to the user.

This thesis clarifies the kinds of projects for which knowledge base systems are best suited, demonstrates their power and makes recommendations on how to efficiently develop them, amongst others by proposing knowledge articulation as a new knowledge acquisition method.

Beknopte samenvatting

Tijdens de laatste decennia is onze samenleving veranderd in een kennismaatschappij, waarin continue innovatie belangrijker dan ooit geworden is. Het zijn niet meer enkel R&D departementen die verantwoordelijk zijn voor die innovatie, maar in toenemende mate wordt deze gedragen door andere experten binnen een bedrijf. In deze thesis onderzoeken we hoe expertkennis gebruikt kan worden om krachtige systemen te ontwikkelen, en hoe deze systemen vorm moeten krijgen om de kennis van experten maximaal te benutten.

In navolging van Liebowitz bekijken we expertkennis op drie niveaus: i) het niveau van de domeinkennis, dit is de feitelijke kennis over het domein, ii) het niveau van inferentiekennis. Dit is de kennis om een kleine, welomlijnde taak te kunnen uitvoeren. iii) het niveau van de taakkennis, dit is de kennis die nodig is om een complexe taak te kunnen uitvoeren, door ze te ontleden in kleinere sub-taken en de correcte oplossingsmethoden te selecteren [90]. Het onderscheid tussen deze soorten kennis kan makkelijk afgestemd worden op het onderscheid tussen domeinkennis en inferentietaken dat gemaakt wordt binnen het kennisbankparadigma. In dit onderscheid zijn de eenvoudigste inferentietaken een voorbeeld van inferentiekennis, terwijl het selecteren van de juiste inferentietaak -of een combinatie van inferentietaken- om een probleem op te lossen, een voorbeeld is van taakkennis.

Het verzamelen van deze verschillende soorten kennis om een kennisbanksysteem te bouwen, vereist verschillende kennisverwervingsactiviteiten. Met behulp van zes gevalstudies ontwerpen we een gedeeltelijke classificatie van beslissingsondersteunende systemen. Daarenboven doen we aanbevelingen voor de kennisverwerving voor elk van de systemen.

De eerste categorie van systemen is juridische toepassingen. Deze systemen ondersteunen gebruikers in het maken van correcte beslissingen op basis van de wet. Correctheid en verklaarbaarheid van de resultaten zijn uiteraard belangrijke eisen van juridische toepassingen. In tegenstelling tot wat vaak

gedacht wordt, wordt het omgaan met wetgeving niet voornamelijk gekenmerkt door deductief redeneren, maar meestal door redeneren met uitzonderingen en onvolledige informatie. Dit heeft een invloed op de ontwikkeling en configuratie van de toepassing, die een hoge mate van interactiviteit moet vertonen. Productconfiguratiesystemen zijn de tweede soort toepassingen. Ze beperken zich niet tot het ontwerpen van tastbare producten, maar kunnen ook worden gebruikt om immateriële artefacten te ontwikkelen, zoals een cursusprogramma of communicatiesjablonen. De kennis van domeinexperten is onmisbaar bij de ontwikkeling van een configurator, en een configuratieproject moet er expliciet op gericht zijn om domein experten te betrekken. De derde categorie van toepassingen is de categorie van planningstoepassingen. De wetenschappelijke kennis over personeelsplanning is enorm, maar in vergelijking daarmee is het gebruik van deze kennis in praktijk beperkt. In deze thesis vinden we dat niet enkel de kwaliteit van de planning belangrijk is, maar ook de sociale rol die planners binnen het bedrijf innemen. De laatste categorie is een overkoepelende categorie van toepassingen voor derden. Hierbij beschikt het bedrijf dat de toepassing ontwikkelt niet over alle kennis. Een meer of minder belangrijk deel van de kennis is enkel beschikbaar voor de derde partij – vaak een externe klant die de toepassing gebruikt.

Dit proefschrift licht toe voor welk soort projecten kennisbanksystemen het meest geschikt zijn en doet aanbevelingen over hoe ze efficiënt ontwikkeld kunnen worden, ondermeer door kennisarticulatie voor te stellen als nieuwe kenningsverwervingsmethode.

List of Abbreviations

AI Artificial Intelligence. 1, 3

ASP Answer Set Programming. 175

ATO Assemble-To-Order. 86, 87, 93

cDMN Constraint Decision Model and Notation. 9, 27, 31, 33–35, 61, 66, 71, 72, 74, 75, 84, 94, 96, 99, 106, 107, 145, 149, 152, 171, 172, 176, 179

CNL Controlled Natural Language. 123–125, 135, 142, 152

CODP Customer Order Decoupling Point. 86

CSV Comma Separated Value. 48

CTA Cognitive Task Analysis. 13

DMN Decision Model and Notation. 9, 18–23, 27, 31, 33, 35, 37, 44, 52, 58–61, 63–66, 70, 71, 74, 75, 93, 94, 96, 99, 106, 133–135, 142, 170, 171, 174, 175, 178, 179

DRD Decision Requirements Diagram. 19, 22, 23, 42–44, 52, 58–60, 64, 99, 149, 153, 170, 171, 174, 176

DTAI Declaratieve Talen en Artificiële Intelligentie. 178, 179

ETO Engineer-To-Order. 62, 66, 67, 84, 86, 87, 93, 157, 159, 163, 171, 177, 180

EUR Euro. 130

FEEL Friendly Enough Expression Language. 20

FO(·) Extended First Order Logic. 8, 18, 23, 24, 29, 35, 37, 39, 45–47, 51, 52, 58, 71, 74, 84, 93, 94, 96, 98, 100, 112, 113, 122–125, 131–135, 142, 145, 146, 148, 152–154, 161–163, 171, 172, 174–176, 178, 179

FOL First Order Logic. 23

GBP Great British Pound. 130

GDPR General Data Protection Regulation. 4

GUI Graphical User Interface. 45, 46, 48

IDP Imperative Declarative Programming. 8, 11, 24, 25, 29, 45, 49, 50, 53, 87, 88, 93, 96, 98, 100, 108, 112, 113, 122, 124, 142, 152, 163, 172, 174, 176, 178–180

KB Knowledge Base. 2, 3, 5, 8–10, 23, 25–29, 35, 45, 47, 52, 55, 57, 59, 60, 63, 84, 88, 89, 91–93, 98, 100, 105, 112, 118, 121, 122, 124, 125, 127, 134, 135, 141, 142, 145, 148, 149, 152–154, 156, 157, 160–165, 169–172, 174–176, 180, 181

KBP Knowledge Base Paradigm. 2, 24, 53, 152

KBS Knowledge Base System. 1–8, 11, 25–28, 32, 34, 37, 46, 52, 53, 55, 61, 92, 94, 134, 156, 164, 166–168, 174, 178–181

KRR Knowledge Representation and Reasoning. 178, 179

LIRIS Leuven Institute for Research on Information Systems. 178

MTO Make-To-Order. 62, 63, 66, 84, 86, 87, 93, 94, 171, 179

MTS Make-To-Stock. 86

NL Natural Language. 122–124, 127, 133, 153, 174

NLP Natural Language Processing. 124, 135, 152

OECD Organisation for Economic Co-operation and Development. 1

PCS Product Configuration System. 61–63, 66, 67, 84–91, 93–95, 133, 141, 142, 156, 157, 159, 160, 163–165, 171, 174, 176, 177, 179, 180

S-FEEL Simple Friendly Enough Expression Language. 20, 21, 32, 33

USD United States Dollar. 130

VB Visual Basic. 62, 63

XAI Explainable AI. 4

Contents

Abstract iii

Beknopte samenvatting v

List of Abbreviations ix

List of Symbols xi

Contents xi

List of Figures xv

List of Tables xvii

1 Introduction 1
- 1.1 Knowledge base paradigm . . . 2
  - 1.1.1 Positioning KBSs . . . 3
- 1.2 Knowledge management in KBSs . . . 6
- 1.3 Research question, methodology and contributions . . . 8
- 1.4 Limitations . . . 10
- 1.5 Structure of this dissertation . . . 11

2 State-of-the-art 13
- 2.1 Knowledge representation and reasoning . . . 14
  - 2.1.1 Classical first order logic . . . 14
  - 2.1.2 Decidable fragments of FOL with description logics . . . 15
  - 2.1.3 Logic programming . . . 16
  - 2.1.4 Advancing expressivity with FO(.) . . . 17
  - 2.1.5 The IDP system and its inference tasks . . . 20
  - 2.1.6 KBS : more than just a label . . . 21
  - 2.1.7 Related knowledge base systems . . . 27

2.1.8 Conclusion . . . 29
2.2 Knowledge acquisition . . . 29
2.2.1 Knowledge acquisition and elicitation . . . 30
2.2.2 Decision model and notation . . . 35
2.3 Developing case study research propositions . . . 41

3 Advancing the state-of-the-art 43
3.1 Knowledge articulation . . . 43
3.2 Interactive consultant prototype . . . 46
3.3 cDMN : constraint decision model and notation . . . 48
3.3.1 Constraint tables . . . 49
3.3.2 cDMN and other formalisms . . . 52
3.4 Conclusion . . . 52

4 Case: Calculation of registration duties 55
4.1 Introduction to the case . . . 56
4.2 Requirements . . . 56
4.3 Decision model . . . 57
4.3.1 Followed approach. . . . 58
4.3.2 Result and implementation . . . 61
4.3.3 Evaluation . . . 62
4.4 FO(.) specification . . . 63
4.4.1 Evaluation . . . 65
4.5 Prototype . . . 65
4.5.1 Advanced prototype . . . 66
4.5.2 Evaluation of the prototype . . . 70
4.6 Discussion of case study results . . . 71

5 Legal applications 73
5.1 Characteristics of legal applications . . . 74
5.2 Knowledge in legal applications . . . 75
5.3 Knowledge acquisition in the legal domain . . . 77
5.4 Recommendations to develop legal applications . . . 78
5.5 Conclusion . . . 79

6 Case: Seal design 80
6.1 Introduction . . . 81
6.2 Case 1: DMN-based make-to-order PCS . . . 82
6.2.1 Requirements . . . 82
6.2.2 Knowledge articulation workshops . . . 82
6.3 Case 2: FO(.) based engineer-to-order PCS . . . 85
6.3.1 Requirements . . . 86
6.3.2 Eliciting physical constraints . . . 86

6.3.3 Eliciting preferences . . . 87
6.3.4 cDMN . . . 89
6.3.5 Prototype . . . 94
6.4 Conclusion . . . 104

7 Product configuration systems 105
7.1 Characteristics of PCSs . . . 108
7.1.1 Development of PCSs . . . 109
7.2 Knowledge in PCSs . . . 110
7.3 Knowledge acquisition for PCSs . . . 111
7.3.1 Evaluation of knowledge articulation method . . . 112
7.4 Recommendations . . . 113
7.5 Conclusion . . . 114

8 Case: Doctor scheduling 116
8.1 Introduction . . . 117
8.2 Requirements . . . 118
8.2.1 Schedule requirements elicitation . . . 118
8.3 Modelling the domain . . . 118
8.3.1 DMN and cDMN . . . 119
8.4 FO(·) model . . . 120
8.5 Prototype . . . 123
8.5.1 Schedule prototype . . . 124
8.5.2 Application prototype . . . 124
8.6 Conclusion . . . 126

9 Staff scheduling 128
9.1 Characteristics of scheduling applications . . . 129
9.1.1 Creating a qualitative roster . . . 131
9.1.2 Quality of IDP to create personnel rosters . . . 132
9.2 Knowledge in scheduling applications . . . 133
9.2.1 Creating fair schedules . . . 135
9.3 Recommendations . . . 138

10 Cases: Third party applications 141
10.1 Case 1: Collateral profile management . . . 142
10.1.1 Introduction . . . 142
10.1.2 Requirements . . . 143
10.1.3 A natural language interface . . . 143
10.1.4 The knowledge base system . . . 144
10.1.5 Evaluation . . . 152
10.2 Case 2: Railway communication application . . . 153
10.2.1 Requirements . . . 153

10.2.2 Maintaining overview of templates . . . 153
10.2.3 Creating new templates . . . 155
10.2.4 Deciding between conflicting messages . . . 157
10.2.5 Using a product configuration system . . . 161
10.2.6 Conclusion . . . 162
10.3 Case 3: Employee's time registration . . . 165
10.3.1 Introduction . . . 165
10.3.2 Modelling the rules . . . 166
10.3.3 Evaluation . . . 169
10.4 Generalizing findings . . . 171
10.5 Recommendations . . . 172
10.6 Conclusion . . . 172

11 Knowledge complexity **174**
11.1 Knowledge complexity in expert systems . . . 175
11.1.1 Knowledge complexity of case study applications . . . 176
11.1.2 Comparing complexity scores . . . 179
11.1.3 Evaluation of the complexity measure . . . 180
11.2 Developing an objective complexity measure . . . 180
11.3 Comparing complexity of KBs . . . 183
11.3.1 Other uses of the complexity measure . . . 185
11.4 Conclusion and future work . . . 185

12 Conclusion **186**
12.1 Using a knowledge base system . . . 187
12.2 Selecting an application type . . . 188
12.3 Choosing knowledge acquisition techniques . . . 191
12.4 Future work . . . 195
12.4.1 General research directions . . . 195
12.4.2 Future work per application type . . . 196

13 Valorization **199**
13.1 Product design . . . 202
13.2 Asset selection . . . 202
13.3 Lessons learned from valorization trajectories . . . 203
13.4 Future valorization steps . . . 203

A Interview protocol **205**

Bibliography **209**

Curriculum **221**

List of publications **223**

List of Figures

1.1 Illustration of kinds of knowledge in different application types 8

2.1 Current ICAO weight categories and associated separation minima. 22
2.2 Classification of KA techniques according to the KA stage (adapted from Leu (2016) 32
2.3 Three activities of graphic elicitation by means of diagrams [29]. 34
2.4 Building blocks of a DRD. 37
2.5 Example of a Decision Table with its elements. 38

3.1 Summary of the knowledge articulation method. 45
3.2 Illustration of kinds of knowledge in different application types 46
3.3 Interactive Consultant with explanation inference. 47
3.4 Constraint tables for body material and design type. 50

4.1 DRD of registration duty calculation with abstract (top right) and expanded view. 58
4.2 Comparison between U and F table for tax rate. 60
4.3 Optimisation in OpenRules. 62
4.4 Translation of variables and impact on interface. 64
4.5 Relevance and explanation demonstrated in the new interface. . 67
4.6 Illustration of a dependency graph. 69

6.1 Optimisation framework. 88
6.2 DRD model of component design decisions. 90
6.3 Decision tables for component design 92
6.4 Combination of constraint and decision tables 92
6.5 Comparison between an open and a closed design in the interactive consultant. 96
6.6 Visual representation of when a constraint is deactivated in the Interactive Consultant (detail of upper right corner). 97

6.7 Indication of a constraint that cannot be reactivated in the Interactive Consultant (detail of upper right corner) 98
6.8 Browsing historical products in the finalised prototype (truncated image). 100
6.9 Expanded unified theory of acceptance and use of technology by Turan. 101
6.10 Unified theory of acceptance and use of technology. 101

7.1 Visualisation of CODP in a production chain (adapted from [10]). 107

8.1 Extract of tables from the doctor scheduling model. 119
8.2 Parsing of Excel calendar in IDP structure 121
8.3 Adjusted IC for doctor scheduling with additional reset and propagation buttons 126

9.1 Determining job crafting activities based on first order codes by Kossek et al. [80]. 139

10.1 Creating a CNL sentence with building blocks. 144
10.2 Natural language statement and its interpretations. 145
10.3 Illustration of haircut rules in a table. 148
10.4 Visualization of interactive propagation. 149
10.5 First hit table of investment profile and its Appendix A format . 151
10.6 Visualisation of funnel effect of rule ordering. 151
10.7 Decision table on train sequence 154
10.8 Background information on station type and location. 156
10.9 Visualisation of created templates. 156
10.10 Truncated screenshot of message selection PCS. 163
10.11 Prototype of message creation PCS. 164
10.12 Interval-based modelling. 167
10.13 Fluents and action based modelling. 168
10.14 High-level DRD of time registration problem. 170

11.1 Parse tree of sentence 11.1 in FO(.) 182

12.1 Main steps to decide on knowledge acquisition techniques. . . . 186
12.2 Decision table on the use of a KBS 189
12.3 Decision tables on application type (left) and knowledge acquisition technique (right). 194

13.1 Next valorization steps. 204

List of Tables

2.1 The FO translation Φ_Q of different S-FEEL conditions Q. . . . 39

9.1 Example of a simple cyclic schedule. 129

11.1 Overview of criteria and weights of knowledge complexity according to Meyer and Culey 176
11.2 Knowledge complexity of our registration duty application. . . 176
11.3 Knowledge complexity of an engineer-to-order PCS. 177
11.4 Knowledge complexity of a scheduling applications. 178
11.5 Knowledge complexity of collateral selection application. 178
11.6 Knowledge complexity of a communication template application. 179
11.7 Knowledge complexity of a time registration system. 179
11.8 Overview of complexity scores. 179
11.9 Knowledge complexity of a vocabulary. 182
11.10 Knowledge complexity of different applications. 183

Chapter 1

Introduction

Since the second half of last century our economy is characterised as a knowledge economy, that is according to the OECD an economy "directly based on the production, distribution and use of knowledge and information" [106]. More than material production factors, the access to and investment in intangible factors such as human capital and research drives innovation and economic growth [115]. Whereas the term knowledge economy was originally associated with knowledge intensive and high-technology sectors such as the chemical sector, it has spread to virtually every sector, including service sectors such as health, education, administration, finance, and retail and wholesale sectors [127]. Moreover, knowledge is not limited anymore to scientific research in R&D departments or universities, it is also accrued in learned skills in the workforce [127]. At the same time, the concept of innovation has been enlarged from the production of new goods, to an innovation process in which new ideas are conceived, developed and implemented [50]. Thus, it may concern new products, but also new services or new business models.

Developments in ICT play an important role in this evolution [127]. Besides traditional IT applications such as word processors, spreadsheet or presentation software, the use of Artificial Intelligence (AI) becomes common in different areas. Examples of this are the use of natural language processing in chatbots, automatic video montage, advanced product configurators, or detecting ripe strawberries [24, 2, 112]. In this research we focus on a specific kind of AI systems, viz., Knowledge Base Systems (KBSs). This kind of systems uses expert domain knowledge as its primary source of information. The expert knowledge can be represented in a computer-readable format, such that the application can use it to reason and propose solutions that comply with the

expert knowledge.

1.1 Knowledge base paradigm

A Knowledge Base System (KBS) is a computer program in which application-independent inference rules are separated from application-dependent knowledge [118]. They are an implementation of the Knowledge Base Paradigm (KBP), that advocates a strict separation between domain knowledge (that is centralised in a knowledge base (KB)), and multiple inference tasks that use this domain knowledge to calculate solutions for a multitude of specific problems [33]. Its emergence can be understood in the light of a trend towards modular software, that started with the split of operating system and utilities from data, datacontrol function and knowledge applications [118]. Specific for a KBS is that multiple inference tasks can use the KB to calculate solutions for different kinds of problems: the KBS is multi-inferential [136]. From this follows that the model in the KB needs to be a truthful representation of the domain. This means that for KBSs that aim to be applicable for a broad range of domains, a very expressive modelling language that is able to capture the unique knowledge of every domain, needs to be used.

The domain knowledge is expressed in a computer-readable format. The whole of this formalized domain knowledge is called the knowledge base (KB). The KB contains *declarative* statements about the world. In itself, it is not executable.[1] However, it can be used by different inferences to calculate solutions for different problems. Because the knowledge in a KB is well-structured and not intermingled with parts of procedures, it is easily accessible, well-readable and easy to validate. Moreover, parts of the information can be adapted without affecting other statements about the domain [118]. As a result, the domain specification is well-maintainable. Also, because the KB contains all relevant domain knowledge, this reduces the need for separate documentation.

The inference algorithms are imperatively coded procedures, that use the information from the KB to calculate solutions for multiple problems. They are *independent* from the knowledge domain and the KB. They are also *generic*, in the sense that an algorithm to calculate, e.g., an optimal model for registration duties, can also be used to calculate an optimal model for financial asset selection or product design. The different inference tasks need to be coherent with one-another: if the model expansion inference task is used to create a model of a KB (i.e., to assign a value to all variables), this model should give the output

[1] In this, it is different from logic programming. Logic programming results in executable programs based on first order logic.

"satisfiable" when the model checking inference is used on it. If the different inference tasks are correctly defined semantically, the coherence of the outcomes will result from it. If this is the case, the KB can be used flexibly to solve different kinds of problems – if only the correct inference task is used.

Another advantage of the separation between domain knowledge and inference tasks, is that independent and generic user interfaces in which a number of inference tasks are included, can be developed. We will show that such a generic interface can be used to quickly create prototypes in diverse fields ranging from statutory law, over financial contracts and communication templates to product configuration. Because statements in a declarative knowledge base are modular, it is easy to create a small prototype with only a few sentences. This prototype can be expanded step-by-step until the complete KB is build. The correctness of each sentence can be checked independently.

1.1.1 Positioning KBSs

Before diving into the development of KBSs, it is important to understand what makes them unique in comparison to other IT systems.

Traditional (imperative) IT systems The main difference between KBSs and traditional software, is the separation of concerns of domain knowledge and inference tasks in the latter. The implication is that basic control structures, such as if-then-else or iterations, that are the building blocks of traditional IT systems, are not present in KBSs. The sentences in a KB are modular and can be put in any order. Traditional applications contain only the domain knowledge needed to complete a specific task. The information that is present, is scattered all over the program and blended with process information. Traditional software is focused on executing one or a couple of tasks, but involves high development costs if a new functionality needs to be added. To ensure maintainability of the system, good documentation is necessary, while a KB is for a large part self-explanatory. In spite of these disadvantages, the use of traditional systems can suffice if a limited number of straightforward operations need to be executed efficiently in a static environment. If the purpose is to store and exploit knowledge in the organisation to develop innovative services or products, the use of a KBS offers more possibilities.

Non-KBP declarative programs A KBS is a type of declarative system. Since the first use of predicate logic as programming language in 1974 [81], the developments in this area have been immense, and resulted in a

plethora of declarative paradigms, such as answer set programming, constraint programming, logic programming, etc. [136]. Typically, these paradigms use declarative domain knowledge on which some kind of inference task can be applied. Declarative knowledge refers to knowledge that does not describe *how*' something should be calculated (as is the case with imperative code), but it describes *what* a "solution" is. It is up to the inference tasks to determine how to calculate the actual solutions. For example, a declarative program will not describe how to calculate a schedule ("how"?), but describes the conditions that a good schedule should meet ("what"?). The inference tasks can calculate different solutions (e.g., calculate a schedule, check if an existing schedule satisfies the requirements of a good schedule, etc.). However, from this does not follow that each declarative programs is a KBS as we defined it. A declarative program may be created with the aim of solving one specific task. A KBS is constructed with the aim of flexibly solving multiple tasks [33].

Data-driven AI applications In data-driven AI applications, machine learning algorithms use extensive data-sets to discover correlations between variables. These correlations are then used to make predictions. The estimation of a subcontractor's performance based on past audits, is an example of such predictions [98]. The underlying idea is that knowledge is implicit in the large amount of data in the form of recurring patterns. By analysing the data, this knowledge can be surfaced and used innovatively. To choose between a data-driven and a KB system, following factors need to be taken into account:

- How important is a correct result? Data driven algorithms provide solutions based on statistical correlations. This means that the solution has a certain degree of probability, for which different measurements to estimate correctness exists (e.g. the portion of true positive, false positive and false negative predictions in the precision-recall curve) [112]. The accuracy of machine learning algorithms can be quite high, but is not 100% certain. For situations in which the user needs to be certain of a *correct* result (i.e., correct according to the knowledge formulated in the system), an application based on expert knowledge is needed.
- How dynamic is the environment? A data driven approach works on past data. This means that if something in the environment changes, it may take a while before this shows up in new data. Old data might be dragged along and/or new training might not be possible. In an expert knowledge system, it suffices to adjust the description of the domain to function correctly in the new environment.
- Should the company be able to explain the result? Although recently a lot of effort is devoted to explainable AI (XAI) it is still the case that data

based algorithms work as black boxes, i.e., the end-user does not know why a specific solution is presented [1]. In an expert-knowledge based system, it is possible to trace an outcome back to the rules that have lead to it. For some applications this is an important factor, for example for matters regulated by the GDPR, that determine that persons have the right to know why decisions on their person were taken [129].

- What is the purpose of the overall project? To use the expert knowledge that is usually implicit and stored in expert's heads, it needs to be made explicit and available. This harvesting of knowledge can be an important project goal [57].
- And last but not least: to which kind of information does the company have access? For a data-driven approach, a lot of datapoints needs to be available, for an expert-knowledge approach the company needs to have access to an expert source that possesses this knowledge.

The choice for a data-driven approach or a knowledge base approach is not mutually exclusive: some cases can be addressed by either one of both approaches, and in some cases the combination of both approaches can result in an added advantage. An example of this is the creation of a personnel schedule. As we demonstrated in the case described in Chapter 8, a KBS can be used to describe the characteristics of a solution, e.g., only one shift per day, or the required number of shifts per week. Also preferences (soft constraints) that improve the quality of the schedule, can be modelled in the form of a cost function. The advantage of this approach is that it uses the information that is available in the organisation, to create a schedule. Users can be certain that the hard constraints will not be violated. Another approach is to take past schedules, and try to learn some patterns in it by using a data driven approach. These learned patterns can subsequently be used to create new schedules. The advantage of this approach is that it does not require the –sometimes challenging – knowledge acquisition effort. On the flip side, one cannot be sure that hard constraints will always be satisfied. Changes in the available resources (e.g., a full-time member starts working part-time and a new part-time colleague is hired) requires retraining the model. In the scheduling example a combined approach is also possible. The knowledge base approach can be used to describe hard and soft constraints (preferences), and a data-driven approach can be used to determine the weight of each soft constraint.

In some cases, only a KBS approach can be used. Take for example a product configuration system, such as the one that we describe in Section 6.2. The purpose of a configurator is to guide a user through all possible combinations, so that in the end the designed product meets their needs. For applications in

which the use of a wrong component could be life-threatening, it is important that the user can be sure that the result satisfies all requirements. Using a data-based approach in this case, would result in proposing a product that answers the needs of other clients with a similar question. The two main problems with this are that users do not have any view on how the system defines 'similar', and that not all possible combinations are explored. Hence, they cannot be sure to have the best product.

This is different for online marketing for example. Here, companies typically do not know very well how and which different variables determine the behaviour of an online shopper. To maintain the potential client's focus, the company wants to show more products than what clients are explicitly looking for. A data-driven approach is then useful to categorize shoppers in a typology , and propose additional products based on it. For humans it would be difficult to develop knowledge on how surfing behaviour predicts purchases because of the enormous amount of knowledge, and its complexity. Therefore, in this case it is less suitable to use a KBS.

1.2 Knowledge management in KBSs

The differences between the different kinds of systems are not only limited to their use, but they also impact the development of the system itself. In this research we focus on the knowledge incorporated in KBSs and how to obtain and manage it. To see how the central role of expert knowledge impacts the way KBSs are developed, it is useful to make a distinction between three kinds of knowledge [90, 61]. First, there is *domain knowledge*: this is the knowledge on the domain of interest, such as its concepts, their characteristics and how they relate to one another. This kind of knowledge is sometimes also referred to as system knowledge. Second, there is *inference knowledge* about how to use this domain knowledge to find an answer to a specific problem. This is also called procedural knowledge. Finally, *task knowledge* or strategic knowledge allows the expert to decompose a complex task in less complex sub-tasks and apply suitable inferences and combine outputs to obtain the final answer. It is about knowing a strategy on how to solve a complex problem.

Traditional software applications Traditional software focusses on this last kind of knowledge: every specific task is programmed separately. Hence, task knowledge is the major kind of knowledge necessary for the development of traditional software. The program might call re-usable procedures to calculate a part of the solution. These small procedures contain inference knowledge:

knowledge on how to solve an atomic subtask. In principle, the idea of callable procedures is that they can be used as a black box, and hence require less knowledge on their functioning. Finally, in this kind of applications, domain knowledge is incorporated *ad hoc* and dispersed. It is implicit, in the sense that if a human reader wants to obtain domain knowledge directly from the program code, it requires an substantial analysis effort. The focus on the kind of knowledge during a development project is reflected by the intensity of the colour in the left panel of Figure 1.1 based on Hall [61]. Major importance is attributed to task knowledge, with the least emphasis on domain knowledge. The importance of inference knowledge is situated between these two extremes. For the construction of traditional IT systems the models that are used encompass process models, data models and user diagrams, etc. The actions to be taken to create IT systems are well documented and multiple methodologies and paradigms exist (e.g., waterfall model, scrum, RUP).

Non-KBP declarative programs Domain knowledge still takes a prominent place in non-KBP declarative programs. However, the need for detailed and epistemologically accurate domain description is less prominent compared to KBSs. Because the program is aimed at a single task, also less task knowledge is needed. Therefore, the overall knowledge level is slightly less elaborated than in KBSs. The knowledge acquisition activities are similar to those of KBSs. For the validation of the KB, less focus will be put on validating the content, but more focus will be put on validating the outcomes of the queries.

Data-driven applications In data driven applications, data is of main importance as suggested by the name. Their purpose is to derive knowledge from data by using and combining algorithms. We have adapted the original figure from Hall [61] to reflect this. Users often have no understanding of the content of the algorithms, hence the light color in the inference knowledge box (Figure **??**). Their main importance is selecting the correct algorithm for the task, reflected as the darker color of the task knowledge box. For data driven knowledge, a lot of data are needed. Activities involved there are searching for suitable datasets and data preprocessing (e.g., cleaning, completing, reformatting), data mining and knowledge interpretation [88].

KBSs The fourth panel of Figure 1.1 reflects the relative importance of the different kinds of knowledge in a KBS development project. Of course, a large effort is directed towards gaining domain knowledge. The analysis of task knowledge and its translation into executable inferences is essential to create a convenient KBS. If only the existing generic inference algorithms are used,

their complexity remains hidden from the developer. In KBS the central role of domain knowledge leads to a much more significant position of detailed domain models, such as product models or decision models. For large domains, the acquisition of all relevant knowledge is very challenging, and can lead to a bottleneck in the development of the application [144, 97].

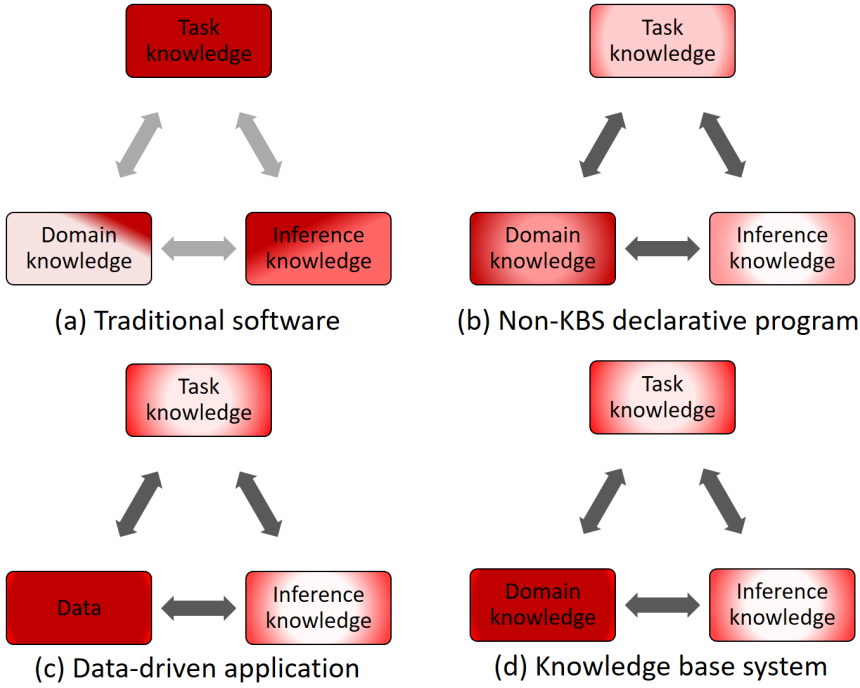


Figure 1.1: Illustration of kinds of knowledge in different application types

1.3 Research question, methodology and contributions

The amount of literature on the broader subject of knowledge acquisition is immense, but concrete guidelines on which technique to use when, are rare. A notable exception is the work of Wei et al. [147] that gives an overview of techniques and some hints on the situations in which it can be used. However, this work does not systematically take into account the specific characteristics of the project at hand, although this is an important factor as will be demonstrated in the remainder of this dissertation.

To fill this gap, we investigate the development of different kinds of KBSs in practice. In this, we focus on different characteristics of each kind of project with regards to the incorporated knowledge, availability of expertise, project goals, etc. We investigate the impact of these factors on the knowledge acquisition technique(s) to use. Moreover, we evaluate the suitability of the selected modelling technique to reflect the nuances and details of the domain knowledge. The aim of this research is to gain an understanding on how one should proceed to acquire knowledge when faced with the challenge of creating a new KBS, and formulate recommendations to this end. In the periphery of this the described case studies also demonstrate how the revealed inference and task knowledge was translated to (a combination of) inference tasks and the resulting application.

We use case study research for this. Yin [150] defines the case study method as "an empirical inquiry that investigates a contemporary phenomenon (the "case") in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident". It is different from an experimental setup in the sense that we do not control the company specific environment in which we operate. This research method is most suitable to gain insight in how KBS project characteristics determine how knowledge is best managed [150]. Because of the real-world context that we cannot control, there are many more relevant variables than the number of data points that we can gather from the limited number of cases. Moreover, multiple sources of evidence should be gathered to support conclusions [150]. The use of statistical methods to generalize findings is typically not very helpful, if possible at all, because of the limited number of cases, the large number of variables, missing reference data, absence of numerical data, ... Instead, what we can do, is to describe factors that contributed to the success of a knowledge acquisition technique in some situations, and possibly find explanations why this technique did not work in other situations. To estimate if the method will work in a new situation or not, the user will need to analyse if these success and fail factors are present in the new situation. This is another form of generalizability, called analytical generalizability [150]. In our research we executed seven case studies. The theoretical premises that underpin our research effort is discussed in Chapter 2.

The main contributions of this research are situated in three areas:

- Advancing KBS in general and the IDP system in particular. To start with, by reading this thesis, the reader will have gained understanding on when the choice for a KBS is recommended, and what the difference are between the development of other systems and KBS. Next, the analysis of the inherent types of knowledge and the empirical description of how to develop a KBS in different cases (subject of Chapters 5 to 10), is new

in KBS research. In the cases that we executed, we were able to validate the claims of maintainability and flexibility of KBSs. Moreover, as this research demonstrates the possibility to model various complex problem domains, it supports the claim that the FO(·) language that we use to create the KB of our KBS, is an expressive enough modelling language. We contributed to the development of new inference algorithms in IDP by decomposing real-life user tasks into inference tasks and pointing to missing inferences. We also propose a first measurement to measure KB complexity.

- Advancing knowledge acquisition. This research shows the importance of attributing a central role to domain experts, and shows different ways in which this can be done. We contributed to the development of methods and tools to support this.
- Creation of a knowledge acquisition framework that recommends the use of specific knowledge acquisition techniques for specific application types. To this end we identify four categories of application areas that are different from a knowledge acquisition point of view: legal applications, scheduling applications, product configuration systems and third party systems. We describe the characteristics of each type and develop a decision model that enables users to decide on the type of their application. This in itself is a contribution, as categorizing helps to reduce the mental complexity of dealing with the objects under consideration [25].

1.4 Limitations

This research does not offer a complete overview of possible knowledge acquisition methods, nor does it address all possible kinds of applications. We started from a variety of cases in which we developed a KBS, and used an appropriate knowledge acquisition method for each case. This means that not for every kind of project guidelines are available. For example, the case in Chapter 9 focusses solely on the scheduling of doctors for the predetermined shifts. We have not focused on the planning of shifts; i.e., how many shifts are needed at what exact moment. In spite of this obvious limitation, some of the insights may still be valuable for other kind of projects. We also did not evaluate all knowledge acquisition methods. We used a sample of methods that seemed appropriate for the cases at hand, but some of the methods have not been touched upon. This means that there remain open research questions, as described in Chapter 4.6.

1.5 Structure of this dissertation

In the next chapter, Chapter 2 we describe the related work of our research. This encompasses some history on the development of logic languages and systems in the field of knowledge representation and reasoning on one hand, and the state-of-the-art in knowledge acquisition on the other hand. We conclude the chapter by describing how the state-of-the-art shapes the lens that we use to design the case study setup.

In Chapter 3 we describe new knowledge acquisition methods and tools that we developed. The first one is the constraint Decision Model and Notation (cDMN), that aims to extend the tabular notation of DMN to constraints. The second method is the knowledge articulation method, that gives domain experts the essential role in knowledge acquisition. We also describe the Interactive Consultant, an interface that can be used for rapid evolutionary prototyping.

The subsequent chapters (Chapters 4 to 10) contain the results of the case study research. Typically we describe the case(s) in depth in one chapter (i.e., Chapters 4, 6, 8 and 10), that is each time followed by a chapter that situates the case in literature and generalises findings (i.e., Chapters 5, 7 and 9). Only for third party applications the cases are described in the same chapter as the generalization of findings (Chapter 10).

We have grouped the cases together in four categories: legal applications (with the case of registration duty calculation), product configuration systems (with two cases of product design), scheduling applications (with the case of doctor scheduling in an emergency service), and third party applications (with cases of collateral selection, communication template creation and time registration calculation). The description of the cases start with an introduction, followed by the knowledge acquisition and modelling techniques that have been used to create a KB. Typically a prototype of the application is discussed afterwards, followed by an evaluation.

The chapters to generalize findings typically start with a definition of the application type, describe the characteristics of the application type, and contain an analysis of the knowledge embedded in the application type, recommended knowledge acquisition methods, general recommendations and a conclusion.

Chapter 11 compares the complexity of the different KBs. As no objective measure existed, we developed our own measure.

Chapter 12 concludes the main findings of this research, and formulates an answer to the research questions. It also contains some indications for future research.

The final chapter of this book (Chapter 13) describes how the results of this research have been and will be valorised.

Chapter 2

State-of-the-art

In the introduction we introduced KBSs, and explained their defining characteristics. In this chapter we will introduce two domains that support the development of KBSs and their use in practice. Knowledge Representation and Reasoning (KRR) is the first domain. As is apparent in the name, the domain of KRR consists of two closely related activities: i) the representation of knowledge and ii) reasoning with the available knowledge. Often these two activities are considered a trade-off: a more expressive knowledge representation formalism renders the reasoning with this knowledge more complicated; and conversely a simple representation allows more efficient reasoning [33]. In the first section (Section 2.1) we sketch the development from classical first order logic to the IDP system and related KBSs.

The second domain is knowledge acquisition. Knowledge acquisition is the process of obtaining and formalizing expert knowledge so we can use it in our KBS. Different techniques and how we can use them, are described in Section 2.2.

Contributions: This chapter situates FO($\cdot$) and the IDP system within the existing KBSs and representations. It gives an overview of knowledge acquisition techniques. From the existing body of literature we develop the lens to look at the case studies in the following chapters.

2.1 Knowledge representation and reasoning

In this section we first look at some of the most reknown logic-based formalisms to model knowledge (2.1.1 - 2.1.4). In 2.1.5 to 2.1.7 we compare some KBSs, starting with an introduction to IDP.

2.1.1 Classical first order logic

FOL is considered the core system of modern logic [49]. In this section we introduce the basic notions of FOL that can be found in a.o. [8]. The **syntax** of FOL determines how it can be correctly written. It consists of a set of well-formed formulas without free variables, also called sentences. A sentence is constructed with logical and non-logical symbols. The logical symbols are connectives, such as "and" ($\wedge$), "or" ($\vee$), "not" ($\neg$), "implication" ($\Leftarrow$), "equality" ($\equiv$), and quantifiers "for all" ($\forall$) and "exists" ($\exists$). Non-logical symbols are determined by the domain of discourse. They are the relations, functions and constants that are used in the domain. A set of sentences could for example take the following form:

$$\forall\ x\ :\ flies(x) \Leftarrow bird(x) \wedge \neg wounded(x)$$

$$\forall\ x\ : bird(x) \Leftarrow crow(x)$$

$$\forall\ x\ : bird(x) \Leftarrow parrot(x)$$

$$\forall\ x\ : parrot(x)$$

The question is: if we have a creature x, which conclusions can we draw based on this set of sentences? The **semantics** determine the meaning of formulas, or what the intention is of the formulas. This meaning is evaluated with a structure that interprets the non-logical symbols. The non-logical symbols are a set of constant, function and relation symbols that together constitute a vocabulary V. A theory T over V consists of a set of sentences that use only the constants, functions and relations of V as non-logical symbols. A structure S over V consists on one hand of the set of all possible values for the elements of V (i.e., the domain), denoted Δ, and on the other hand of an interpretation that assigns a value of Δ to each constant of V, a set Δ_n to n-ary predicates of V and an image $\Delta_n\ \rightarrow\ \Delta$ to n-ary functions of V. A structure S over V is a model M of T over V if each sentence ϕ in T is true in M, also denoted $M\ \models\ \phi$.

Given a well-formed set of FOL sentences with its semantics, we can use it to derive logical conclusions by applying deductive **reasoning**. With deduction new information is logically derived from known information. Based on the known premise(s), new conclusion(s) can be drawn. In the example we know that x is a parrot, so it must be true that x is a bird $(\forall\ x:\ Bird(x))$.

Despite its major importance for logic, FOL has some disadvantages for use in real-life applications. FOL theories are undecidable for deduction, meaning that it is impossible to find a general decision procedure that can unequivocally establish for every theory if a statement follows from it [142]. Moreover, some knowledge cannot be described naturally in FOL [149]. Hence, several extension and variants of FOL have been created.

2.1.2 Decidable fragments of FOL with description logics

DLs are less expressive fragments of FOL that are used to overcome its inherent undecidability. It makes a distinction between general knowledge and instance knowledge represented in T-Boxes and A-Boxes respectively. T-Boxes contain generic statements, such as "all parrots are birds". A-boxes on the other hand, contain assertion knowledge, such as "Bubba is a parrot". DLs assume an open world: the information that is available in the domain Δ, might not contain all information [103]. A fragment uses concepts, roles and individuals; and combines them by using connectives $\sqcap$, $\sqcup$, $\neg$, $\forall$, $\exists$ to express the relations between them in axioms. The concepts in DL are similar with the concepts in FOL, coated with some syntactic sugar. The overview below provided by [142] puts some concepts next to each other.

DL	FOL	
Concept	Unary predicate	
Role	Binary predicate	
Individual	Constant	
$C_1 \sqcap C_2$	$C_1(x) \wedge C_2(x)$	with C_1, C_2 concepts
$C_1 \sqcup C_2$	$C_1(x) \vee C_2(x)$	with C_1, C_2 concepts
$\neg C_1$	$\neg C_1(x)$	with C_1 a concept
$\forall R.C_1$	$\forall y R(y,x) \implies C_1(y)$	with R a role and C_1 a concept
$\exists R.C_1$	$\exists y R(y,x) \wedge C_1(y)$	with R a role and C_1 a concept

Thanks to syntactic restrictions on the formulation of axioms, the DL-fragments are decidable [103]. This, together with the open world concept, makes DL useful for many situations. Sadly, its restricted syntax limits its use in other situations. For example, Motik and Rosati found that relations that do not

have a tree-like structure are impossible to represent in DLs [103]. Other shortcomings are the inability to model exceptions, closed world reasoning and integrity constraints [103].

There exist many varieties of DL, that can be reduced to one of three basic logics: $\mathcal{AL}$ (Attributive Language), $\mathcal{FL}$ (Frame Based Description Language) or $\mathcal{EL}$ (Existential Language). Additional extensions can be added. E.g., $\mathcal{ALC}$ is a commonly used DL that extends $\mathcal{AL}$ with complex concept negation.

2.1.3 Logic programming

From the early 1970's on, the logic programming paradigm emerged with the work of Kowalski on Predicate Logic as a Programming Language [81]. Logical programming is a form of declarative programming that uses its own programming languages that are based on FOL. Typically, these languages use declarative clauses of the form $B_1 \ \ldots \ B_n \ \leftarrow \ A_1 \ \ldots \ A_m$. A clause containing the variables $x_1 \ \ldots \ x_k$ is interpreted as universally quantified as $\forall \, x_1 \ \ldots \ x_k, B_1 \textit{ or } \ldots \textit{ or } B_n \textit{ is implied by } A_1 \textit{ and } \ldots \textit{ and } A_m$. A set of clauses $C_1 \ \ldots \ C_j$ is interpreted as a conjunction $C_1 \textit{ and} \ldots \textit{ and } C_j$ [81]. Different kinds of semantics are possible for logic programs. At the time Kowalski, together with Van Emden, described operational and fixpoint semantics for logic programs [83]. Throughout the years, different semantics have been developed, including completion semantics also known as Clark completion [27], stable set semantics [55] and well-founded semantics [135]. The different semantics use different forms of reasoning. Monotonic reasoning refers to a form of reasoning in which the conclusions stay valid, even if new information is added. This kind of reasoning ensures the modularity of a KB, because new information can be added without invalidating the existing conclusions. Well-founded semantics is a form of monotonic reasoning. It uses three truth-values: true, false and unknown [135]. The values "true" and "false" are only assigned if the values are justified. Hence, adding new information cannot undo this justification. Non-monotonic reasoning is the counterpart of monotonic reasoning. With this kind of reasoning the conclusions of an existing specification may change if additional information is added. Stable model semantics is an example of non-monotonic reasoning. It is a two-valued semantic that uses negation as failure (NAF) [55]. This means that the "true" value is assigned based on the specification, and everything that is not "true", automatically gets assigned the "false" value. Therefore, atoms might change from "false" to "true" when additional information is added.

2.1.4 Advancing expressivity with FO(.)

In contrast with logic programming that uses predicate logic as programming language, FO(·) extends FOL as modelling language. This means that a specification in FO(·) is not an executable program, but it is merely a bag of information. The (·) stands for the different extensions that have been made to FOL : FO(Agg) for aggregates that have been added, FO(ID) for inductive definitions, FO(PF) for partial functions and FO(T) for types [31].

Inductive definitions Much of our knowledge in a domain is definitional knowledge. For example, the registration duties that are calculated in the case of Chapter 4 can be seen as a definition of the registration duty in terms of the registration type. A definition has the specific property that it contains all necessary and sufficient conditions for the defined concept to be true. A special kind of definitions are inductive definitions, that contain constructive information of how a new instance of a relation is defined in terms of existing instances of a relation. A typical example is the *ancestor* relation: your parent is your ancestor, but also an ancestor of your ancestor, is your ancestor. This kind of definition cannot be expressed in FOL. FO(ID) adds the definition syntax with its related semantics to FOL. A definition is a set of rules that intends to define the defined predicate(s) in the head in terms of the open predicates in the body. Definitions define the truth of the defined predicates. They follow the following syntax:

$$\left\{\begin{array}{l} p \leftarrow \psi_1. \\ p \leftarrow \psi_2. \end{array}\right\}$$

The formal development of FO(ID) was proposed by Denecker in his ICLP2000 paper "Extending Classical Logic with Inductive Definitions" [32] that won the Alan Robinson 20 Year Test Of Time Award at ICLP2020. Its meaning is established by well-founded semantics. Next to the semantics to interpret inductive definitions in FO(ID), this paper also describes some characteristics of definitions. To start with, a definition completely defines a concept. In the example of the registration duty, the conditions for every possible registration duty needs to be defined in the definition, and conversely each registration type needs to have a corresponding registration duty. Second, inductive definitions in FO(ID) are equivalence preserving with regards to the joining of rules. A definition with two rules $p(x) \leftarrow F_1$ and $p(x) \leftarrow F_2$ is logically equivalent to $p(x) \leftarrow F_1 \vee F_2$. This has the advantage that different conditions of a definition can be put in different rules, which makes it easier to read. Note

that without the specific definition-notation, one is forced to use an equivalence containing all conditions for non-inductive definitions. It is impossible to formalise inductive definitions at all. Definitions are not only well readable, but also elaboration tolerant: a definition of a concept can be easily expanded as human understanding evolves [96]. Adding rules inside a definition is a non-monotonic operation, meaning that the conclusion from the definition may change if new information is added. The definition as a whole, on the other hand, is a monotone module, as are axioms. It is the monoticity of axioms and definitions that results in the much-wanted modularity of the represented knowledge [32].

Aggregates are introduced with FO(Agg) to calculate the cardinality, minimum, maximum and sum of a given set. An aggregate is of the form

$$Agg\{v_1..v_n : \phi : t\}$$

with Agg the name of the aggregate function (sum, min, max, count), $v_1 \ \ldots \ v_n$ the variables used in ϕ and t, the formula ϕ to select the objects over which the aggregate should be applied and the term t that needs to be added up or optimised. For the cardinality aggregate, the term t is absent as simply 1 is added when ϕ is satisfied. Denecker, Pelov and Bruynooghe developed well-founded and stable semantics for programs with aggregates using approximation theory [35]. Approximation theory will transform the original program P in a truth function T_p such that the stable, well-founded and completion semantics of P and T_p are the same. This paper of Denecker et. al. [35] was recently awarded the Alan Robinson 20 Year Test Of Time Award at ICLP2021.

Partial functions are not allowed in FOL, although they frequently occur in practice. Often, only a partial interpretation of a function is known, and the purpose is to find the missing values. Or, for some values there does not exist a function value. For example the function $MarriedTo(x) = y$ is not defined for bachelors. A partial function can be declared explicitly in the vocabulary, by adding the word "partial" in front of the function. The vocabulary for the "MarriedTo" example hence looks as follows:

$$Type\ Person$$

$$partial\ MarriedTo(Person) : Person$$

The semantic interpretation of a partial function F depends on the context in which it occurs. If it occurs in a sentence and with an even number of

negations, or it occurs in the body of a rule with an uneven number of negations, $P(..., F(x), ...)$ is interpreted as $\forall\ y(F(x) = y \Rightarrow P(..., y, ...))$. In all other situations $P(..., F(x), ...)$ is interpreted as $\exists\ (F(x) = y \wedge P(..., y, ...))$ [85]. Decat et.al. give the example of the predicate $White(Unicorn)$. in the first case it is interpreted as $forall\ x : \ Unicorn = x \ \Rightarrow \ White(x)$: if the unicorn exists, than it is white. In the second context it is interpreted as $\exists\, x : \ Unicorn = x \ \wedge \ White(x)$: the unicorn exists and is white [31].

Using types in a knowledge representation language allows a more concise representation of the world, with the additional advantage that mistakes are easier to detect. For example, in the case of registration duty calculation (Chapter 4) two viewpoints can be taken. The registration duty is due on the purchase amount of the property. An appropriate function would be $RegistrationDuty(Property) = Amount$. However, in the end it is of course always a person who needs to pay the duty. If a couple buys a house in equal shares, the registration duty is also evenly divided. If one buyer previously bought and sold another property, in some cases the registration duty paid on the former property can be deducted from its share of the new registration duty. Hence, it also makes sense to calculate the registration duty per buyer. $RegistrationDuty(Buyer) = Amount$ could be a good function to represent this. Hence, the use of typing disambiguates between different meanings, and makes the language more expressive. In practice, an FO(·) KB contains a vocabulary that represents the ontology of the domain. In this vocabulary the types are declared, and the n-ary relations and functions with n > 0 contain the types of their arguments. When used in the theory, arguments need to be of the correct type. Also, a domain is associated with each type. This is typically done in the structure part of the KB, although some domains can be assigned when declaring the type in the vocabulary.

Arithmetic is indispensable to represent certain kinds of knowledge, for example to calculate distances, amounts, and even in logic puzzles. FO(·) offers the possibility to use arithmetic over integers, more specifically addition, subtraction, negation, multiplication, division, modulo and absolute value.

The extensions allow to create complex formulas in the theory. This is indispensable to be able to create an "objective" (i.e., independent of the specific task to complete) description of in the domain in the context of a KBS that can be used to solve a multitude of problems.

Example of FO(.) knowledge base.

We demonstrate the syntax and principles of a FO(·) KB on a toy graph coloring problem with three nodes and three colors, in which connected nodes need to have a different color. A KB in FO(·) consists of three sections: a vocabulary V which contains the ontology of the domain, a theory T which contains formulas to describe the relation between different elements of the ontology, and a structure I_p which assigns an interpretation to some of the symbols in V. We also say that the structure contains a partial interpretation of the vocabulary. The *vocabulary* contains the types from our domain of interest, and n-ary predicates (relations) and functions that contain these concepts. Predicates and functions without arguments are booleans and constants respectively. To compute a coloring of a graph, we consider the typed vocabulary consisting of types $Node$ and $Color$, the predicate $Edge(Node, Node)$ and the function $Color : Node \rightarrow Color$. We can then express the required domain knowledge by the theory T that consists of the following formula, stating that two connected nodes must have a different color:

$$\forall x\ y : Edge(x, y) \Rightarrow ColorOf(x) \neq ColorOf(y).$$

The structure I_p over vocabulary V typically contains the domain D and an interpretation of some, but not all, of the symbols in V, e.g.,

$$Node^{I_p} = \{A, B, C\}$$

$$Color^{I_p} = \{R, G, B\}$$

$$Edge^{I_p} = \{(A, B), (B, C), (A, C)\}$$

2.1.5 The IDP system and its inference tasks

In this research, we have chosen to make use of the IDP system, a state-of-the-art implementation of the KBP[149, 31]. The IDP system seeks to overcome the undecidability of FOL by using less demanding but practical forms of inference [33]. The inference tasks should return a correct solution, regardless of the domain described inside the KB. In other words, they need to be generic. IDP stands for *imperative declarative programming*, which refers to the declarative FO(·) language used to create the KB, and the imperative Lua code used for calling the inference algorithms. The latest version is IDP3.[1] IDP uses a ground-and-solve strategy and an underlying minisat solver [45, 31]. Grounding means that in a first step the known values from the structure are filled in, and

[1] see IDP homepage at *https://dtai.cs.kuleuven.be/software/idp*)

in the next step the open values are solved. Recently, Microsoft's Z3 satisfiability modulo theories solver is used as underlying solver for the IDP-Z3 version under development.[2]

IDP allows a variety of different inference tasks to be applied to a KB. Probably the most used inference task is that of *model expansion*: given an interpretation I for part of the vocabulary of a given theory T, the system is asked to compute an interpretation J for the remaining part of the vocabulary such that the two interpretations together satisfy the theory, i.e., $I \cup J \models T$. The IDP systems contains a number of different algorithms and techniques from domains such as Answer Set Programming, Constraint Programming and Logic Programming that allow it to efficiently implement this flexible behaviour.

In addition to model expansion, IDP also allows other inference tasks to be performed on a given theory. For instance, it also supports the task of *optimising* the value of a given term. If we consider the term $\#\{x : Color(x) = R\}$, which represents the cardinality of the set of nodes that are red, we can ask IDP to compute not just any model expansion of the theory T and a particular input interpretation I, but the model expansion J for which the interpretation t^J of this term t is minimal. In this way, we compute the graph coloring in which as few nodes as possible are colored red.

As a final example, we mention the inference task of *propagation*: given an input interpretation I and a theory T, compute all of the consequences of I according to T, formally defined as the set of all atomic properties that are either true in all model expansions of T in I, or false in all of them. Informally this means that, given an input interpretation I, forced and forbidden values are derived and uncertain values are reported.

2.1.6 KBS : more than just a label

The fact that we use a KBS puts some requirements on the expressivity of the modelling language. As it should "stand on its own", irrespective of the task that one wants to solve with the system, a specification needs to be declaratively correct. Hence, the modelling language should be expressive enough to model the domain at hand. We demonstrate this with the implementation of air traffic regulation guidelines in two systems with different formalisms: FO($\cdot$) and PSOA RuleML as described in our 2019 RuleML paper titled "Aligning, Interoperating, and Co-executing Air Traffic Control Rules across PSOA RuleML and IDP" [7]. The paper under supervision of Joost Vennekens was joint work with Theodoros Mitsikas, Sofia Almpani, Petros Stefaneas, Panayiotis Frangos, Iakovos Ouranos,

[2] https://idp-z3.be

Harold Boley. The co-authors provided knowledge on the specific case and the PSOA RuleML input.

Air Traffic Control Regulation

Collision prevention of aircraft is realized in the Air Traffic Control (ATC) regulations by ensuring a minimum distance between aircraft, a concept also called *separation minimum*. Separation of aircraft serves an additional role, which is the avoidance of wake turbulence. The separation minimum is defined for aircraft pairs depending on their wake turbulence category. ATC are issued by the Federal Aviation Administration (FAA) and the International Civil Aviation Organization (ICAO) The current FAA and ICAO regulations categorize aircraft according to their Maximum Take-Off Weight/Mass (MTOW/MTOM). MTOW/MTOM represent the wake turbulence of the leading aircraft, as well as how much a following aircraft is affected by the wake turbulence of the leader. Both agencies are in the process of a wake turbulence recategorization (RECAT), which recategorizes aircraft in six categories, taking into account the wingspan as an additional parameter.

For example, ICAO discerns four categories: Light (MTOM of 7000 kg or less), Medium (MTOM of greater than 7000 kg, but less than 136000 kg), Heavy (MTOM of 136000 kg or greater), and Super (a separate designation that currently only refers to the Airbus A380 with MTOM 575000 kg) [72, 86]. Table 2.1 shows the required minimal distance (in Nautical Miles (NM)) between leader and follower plains of different categories. The Minimum Radar Separation (MRS), is 3 NM or 2.5 NM depending on operational conditions unrelated to wake turbulence (e.g. visibility) [86].

Figure 2.1: Current ICAO weight categories and associated separation minima.

		Follower			
		Super	Heavy	Medium	Light
Leader	Super	MRS	6	7	8
	Heavy	MRS	4	5	6
	Medium	MRS	MRS	MRS	5
	Light	MRS	MRS	MRS	MRS

PSOA RuleML

The Rule Modelling Language, or RuleML in short, is a unifying system that can be used to exchange knowledge bases and queries between rule based systems, by proposing a schema-defined syntax and formal semantics [74]. PSOA stands for Positional-Slotted Object-Applicative RuleML [13]. The aim of PSOA is to develop a logical language that allows to describe domains in an object-oriented way. An atom $Aircraft(: mtow \rightarrow 9300.0)$ means that there exists an object of type Aircraft with attribute mtow that has value 9300.0. It is also possible to give this object a name by writing $be91\# : Aircraft(: mtow \rightarrow 9300.0)$. These are examples of ground atoms, i.e., atoms without variables. It is also possible to use variables in PSOA. They can be recognized by the prefix '?', as in $Aircraft(: mtow \rightarrow ?x)$. Variables appear in rules. A rule in PSOA is a universal quantified implication with the conclusion at the left hand of the condition. A conclusion is a PSOA atom, a conclusion can be a PSOA atom or a conjunction of PSOA atoms. The quantification happens with the use of the `Forall` wrapper, the implication uses `:-` , and the conjunction happens with the `And`-prefix.

The reference implementation for deduction in PSOA RuleML is the open-source framework system PSOATransRun, currently in Version 1.5.1 [3].

The fragment below represents the regulations according to the ICAO regulations, modelled in PSOA RuleML.

```
Forall ?a ?w (
  :AircraftIcaoCategory(?a icao:Light) :-
    And(?a#:Aircraft(:mtom->?w)
        math:lessEq(?w 7000))   )

Forall ?a ?w (
  :AircraftIcaoCategory(?a icao:Medium) :-
    And(?a#:Aircraft(:mtom->?w)
        math:greaterThan(?w 7000)
        math:lessThan(?w 136000))   )

Forall ?a ?w (
  :AircraftIcaoCategory(?a icao:Heavy) :-
    And(?a#:Aircraft(:mtom->?w)
        math:greaterEq(?w 136000)
        not:Naf(:AircraftIcaoCategory(?a icao:Super))
```

[3] `https://psoa.ruleml.org/transrun/1.5.1/local/`

```
    )
)

:AircraftIcaoCategory(:a388 icao:Super)
:AircraftIcaoCategory(:a38f icao:Super)

%***********ICAO Separation example***********

Forall ?l ?f (
    :icaoSeparation(:leader->?l
                    :follower->?f
                    :miles->8) :-
    And(:AircraftIcaoCategory(?l icao:Super)
        :AircraftIcaoCategory(?f icao:Light))
    )

%***********Sample Aircraft Facts***********

:be91#:Aircraft(:mtom->4218.41
                :mtow->9300.0
                :wingspan->45.92
                :appSpeed->100.0)

:a388#:Aircraft(:mtom->575000.0
                :mtow->1267658.0
                :wingspan->261.65
                :appSpeed->145.0)
```

FO(.)

We have created in FO(·) a specification that is classically equivalent with the PSOA specification, in the sense that they both have the same class of possible worlds: an interpretation (called a structure in IDP) W satisfies the PSOA fragment P if and only if it satisfies the IDP fragment I: $W \models P \Leftrightarrow W \models I$.

```
vocabulary V {
    type Mtom isa int
    type Aircraft isa string
    MTOM(Aircraft,Mtom)
        ... }
theory T:V{
```

```
    !a[Aircraft] w[Mtom]:
    AircraftIcaoCategory(a, Light) <=
        MTOM(a,w)
            & w =< 7000.

    !a[Aircraft] w[Mtom]:
    AircraftIcaoCategory(a, Medium) <=
        MTOM(a,w)
                & 7000 < w
                & w < 136000.

    !a[Aircraft] w[Mtom]:
    AircraftIcaoCategory(a, Heavy) <=
        MTOM(a,w)
                & 136000 =< w
                & a ~= a388
                & a ~= a38f.


    AircraftIcaoCategory("a388", Super).
    AircraftIcaoCategory("a38f", Super).

    //***********ICAO Separation Example**********

    !l[Leader],f[Follower]:

    IcaoSeparation(l, f) = 8 <=


        AircraftIcaoCategory(l, Super)
        & AircraftIcaoCategory(f, Light).
    }

structure S1 : V {
    //***********specific value assignments**********
    Leader = {a388}
    Follower = {be91}
    ...
    //***********aircraft data**********
    MTOM = {be9l, 4218; a388, 575000}
    MTOW = {be9l, 9300; a388, 1267658}
    WingSpan = {be9l, 45; a388, 261}
```

```
AppSpeed = {be91, 100; a388, 145}      }
```

Semantics and truthful representation

As written above, both PSOA and FO(·) represent the categorization of aircraft as a set of implications: the `:-` symbol of PSOA and the `<=` symbol of FO(·) both denote the material implication of FOL. In other words, an interpretation W is a model of an implication `F :- G` in PSOA, or of `F <= G` in FO(·), if and only if G holds in W or F is false in W. Accordingly, the class of models of the above FO(·) / PSOA specification is quite large, since every superset of a model is again a model; e.g., there are models in which the same aircraft belongs to all four categories at the same time.

The existence of these "extra" models is not considered a problem for PSOA, as long as only the specific inference task for categorization is used. Because this system uses the KB by means of the inference task of *query answering*, it looks for properties that hold in *all* models of the specification. However, it is not a correct declarative representation of the available knowledge.

The IDP system offers the same inference task, allowing the above FO(·) specification to be used in precisely the same way (and, because of the equivalence of the two specifications, producing identical results). However, the idiomatic use of IDP as a KBS requires that the same KB should be usable by different inference tasks. To guarantee that different inference tasks produce correct results, it is crucial that the KB is constructed such that its models correspond one-to-one to possible worlds in reality. The above specification obviously does not have this property and therefore produces correct results *only* when the inference task of query answering is applied to it. When applying, e.g., the inference task of *model expansion*, we will obtain erroneous results, in which an aircraft is assigned some category even though it does not satisfy the condition for belonging to this category (such as `:be91` belonging to the category `Heavy`).

The more idiomatic way of representing the above knowledge in FO(·) is by means of a *definition*, replacing each of the material implications $\Longrightarrow$ by FO(·)'s *definitional implication* symbol $\leftarrow$. Such a definitional implication entails the material implication (i.e., if F $\leftarrow$ G then also F $\Leftarrow$ G), but in addition, it also implies that F can *only* be true if there is at least one rule of the form F $\leftarrow$ G such that G is true. As mentioned above, such a set of definitional implications is interpreted under the well-founded semantics. Since the above specification is not recursive, this means that it is equivalent to its Clark's completion. In other words, it states that an aircraft belongs to a certain category if and only if it satisfies the corresponding condition. Therefore, the definitional FO(·)

specification only has a single model, in which each aircraft is assigned a single category, namely that whose condition it satisfies.

In [7] we demonstrate how the use of definitions allowed us to detect inconsistencies in the issued regulation prior to adding the structure with the list of existing aircraft. The detection of these inconsistencies in PSOA RuleML required a witness case of an aircraft that did not have a category, or that belonged to two categories [7]. Moreover, we also demonstrate how the correct KB could also be used for other inferences, such as calculating the optimal landing order in case more than two aircraft arrive at the same time [7].

2.1.7 Related knowledge base systems

IDP has been explicitly designed following the KBP. In this section we examine the existence of other KBSs. Before we dive in to different systems, we bring to mind our definition of a KBS following the KBP. Within a KBS a strict distinction is made between a declarative knowledge base that contains domain knowledge, and the inference tasks that use this knowledge to calculate solutions for different problems. Essential is the existence of multiple, generic inference tasks. Therefore it requires a modelling language whose expressivity matches the complexity of the domain. It is important to stress this, as the name "knowledge base system" is widely used in different contexts. For example, in a paper on the integration on a KBS and a natural language interface mentions that "KBSs which use traditional databases are not effective [...]" [78]. Solely from the idea that the underlying knowledge may be as minimal as a simple database it is apparent that this interpretation of KBS does not coincide with our definition. Typically, this kind of papers do not give a definition of what a KBS is.

The case of the ATC regulations already demonstrated that declarative systems that centralise the domain knowledge in a KB, are not necessarily KBSs that follow the KBP. To qualify as a KBS, the KB needs to be separated from the inference tasks, the KB needs to be formalised in such way that it truthfully represents the domain regardless of the final purpose of the application, and generic inference tasks exist/can be developed to execute multiple tasks on the same KB. The PSOATransRun system does not meet these criteria as it is based on query satisfaction only, and specific queries need to be developed for each new task.

We discussed DLs as decidable fragments of FOL. There exist several systems that use DLs, such as FaCT, RACER, CEL and KAON. The most recent version of FaCT is called FaCT++. Tsarkov and Horrocks describe the system in their paper "FaCT++ Description Logic Reasoner: System Description" [130]. In this

paper they describe one inference task, i.e., query satisfiability checking. The main research focus seems to be on optimising the performance of this inference, rather than on developing new inference tasks. Moreover, in the introduction to DLs we mentioned that DLs are decidable because of the restricted syntax of the language. As examples a.o. the expression of non-tree-like relations and reasoning with closed world assumption were given. Therefore it seems unlikely that DL-based systems will evolve into widely-applicable KBSs.

Seipel et.al. studied Domain-Specific Languages (DSLs) as an approach to model expert knowledge in a specific application area [119]. They note the distinction between modelling languages and programming languages, and specify that the former are not executable, whereas the latter are. Because much expert knowledge is modelled in a modelling language, they propose to use declarative DSLs that contain if-then rules with the DDbase system [119]. Although their proposal resembles the basics of the KBP, i.e., the distinction between a declarative KB and the inference task, it also shows important differences. The declarative expert knowledge can only be expressed in the if-then format, which strongly limits the expressive power of the DSL. Moreover, the only inference task executed by DDbase is forward chaining (sometimes also called the bottom-up approach) [119]. This means that starting from the facts, more and more facts are derived, to finally come to a conclusion. Hence, the system misses the flexibility that is the result from the use of generic inference tasks with an expressive KB.

Perhaps the most KBS-like system is the ASP-based Potassco system developed at the University of Potsdam. Potassco has commercial applications in different domains [153]. It is an efficient application with a modelling language that allows to declaratively model the problem in ASP [54]. In contrast to FO($\cdot$) this language is not based on FOL. Potassco uses stable model semantics, meaning that anything that is not explicitiely made true, is rendered false. Although ASP in theory only allows one form of reasoning (i.e., answer set calculation), the last few years the development of Potassco has focussed on mimicking other forms of inference [54]. Model expansion, optimisation, model checking, propagation,... are forms of inference that are possible in Potassco. However, their interpretation is not necessarily the same as their interpretation in IDP because of the two-valued truth-values used in Potassco. Recently a translator from FO($\cdot$) to ASP has been developed [134]. This could stimulate further development in both formalisms and their implementations.

2.1.8 Conclusion

The field of knowledge representation and reasoning is confronted with a trade-off between the expressivity of a language and the reasoning power of a system using it. We started with describing the basics of modern logic on the form of FOL. Because FOL is undecidable, DLs use only fragments of the logic, that are less expressive, but decidable. The developers of the IDP system have opted for another approach. By adhering to the KBP the expressivity of the language and the reasoning power of the system became separate (but interlinked) research questions. The expressivity of FOL was extended in the new language FO($\cdot$). The reasoning power of the system was advanced by using approximation theory for well-founded semantics. As far as we know, the IDP system is unique in its kind, although with recent developments the ASP-based system Potassco might also be described as KBS. A profound evaluation of the Potassco system is expedient to confirm this point of view.

2.2 Knowledge acquisition

The main source of knowledge for KBSs is expert knowledge. In many companies, multiple experts with partially overlapping areas of expertise are involved in the creation of similar products. This means that the selection of the expert to be involved becomes an important topic at the outset of the project. There exists a rich body of knowledge on the difference between experts and novices in different fields of science [117, 53, 125]. The often-used scale of Hoffman [67] discerns (in increasing order of development) initiate, novice, apprentice, journeyman, expert and master development levels.

From research in multiple fields of practice, it emerges that novices use one reasoning style (that is, linear, forward reasoning) and are overly relying on existing procedures and presented data [117, 53, 125]. Experts on the other hand, switch between multiple forms of reasoning, dare to question data and take also seemingly irrelevant data into account. Specifically in design, novice users are found to solely use backwards and deductive reasoning, while experts use more often forward reasoning, but they combine it with backwards reasoning in complex cases [4]. Novice users are found to use a trial and error design pattern, in which decisions are taken and evaluated post hoc. This way of working stands in contrast to the experts' way of working. They use a *priori* design pattern, in which they evaluate the design *before* putting it to practice. When designing, they refer to historical designs to understand why some component has been used, to find similar problems and how they were resolved, to ensure consistency between designs, ... [4]. Moreover, experts look at the whole and

coherence of issues, while novices treat issues sequentially and have difficulty to differentiate between more and less important issues [4].

Different methods can be used and combined together to establish proficiency to determine the "pool" of experts that can be used for knowledge acquisition, e.g., career interviews, possessing professional licences, performance measurement, or social interaction analysis [66]. The final selection of the expert that will be involved in the project, will depend on other factors such as the willingness to cooperate and the availability of the expert. The latter is typically an issue, as experts are often over-questioned in an organisation.

However, in many cases having multiple experts providing input may provide considerable added value:

- More and better information and solutions thanks to the broader knowledge [58].
- To cover the full width of the domain in case it is very broad and the expertise of one expert does not fully cover it [58].
- In semi- or ill-structured domains, to agree on relevant information and find creative solutions [58].
- To obtain multiple viewpoints as input to construct a single, aligned view.
- To elicit implicit knowledge through interaction [28].

2.2.1 Knowledge acquisition and elicitation

Very broadly defined, knowledge acquisition is about obtaining insight in the (mental) processes that underpin the observable outcomes in companies [88]. This is generally understood as a process that starts at available knowledge sources, and ends with computer-readable models of this knowledge. Many authors do not give a definition of knowledge acquisition, but refer immediately to the activities that are part of the knowledge acquisition process. However, there is no unanimously accepted vision on the exact activities and the number of steps needed to get to the end result. Hua [70] for example, defines knowledge acquisition as the elicitation, collection, analysis, modelling and validation of knowledge. Other authors only consider three different activities. In this dissertation we follow the definition of Leu et al.[88], who discern three steps in the knowledge acquisition process: i) knowledge elicitation, or the formulation of knowledge by domain experts, ii) knowledge explication, or the analysis and interpretation of the elicited knowledge by a knowledge engineer, and iii)

knowledge formalisation, or the modelling of the explicated knowledge in formal models by the knowledge engineer.

For knowledge elicitation, a plethora of techniques exist (this would go in the hundreds according to [88]!), ranging from interviews over observation to protocol analysis and much more. There exist many differences between them, as well as a variety of ways to classify them in comprehensible categories. One of the commonly known classifications is based on the differential access hypothesis. This hypothesis states that the elicitation method determines the kind of knowledge one obtains from it [69]. Hence, methods can be classified according to their information output. One explanation for this is that the elicitation method impacts the reasoning strategy of the expert, who as a result focuses more or less on specific aspects. Another argument states that some knowledge is implicit or tacit, and experts will not be able to verbalise it, unless an adapted elicitation method is used [69]. Although a lot of research has been devoted to this hypothesis, there are no conclusive results, and attention to the topic diluted in favour of the idea of differential utility and relative efficiency [66]. The relative efficiency of a method is the 'information yield' of the method, expressed in the number of useful propositions obtained per total task minute of the elicitation exercise [68]. The idea of differential utility stems from the idea that each method has its (dis)advantages, that may display a better fit with the project at hand [66]. Moreover, the idea of opportunistically combining different methods in one project gains ground [66].

Other classification methods do not solely focus on knowledge elicitation techniques, but on knowledge acquisition techniques in general, and classify them according to the knowledge acquisition phase they support [88]. Figure 2.2 shows an extract of human agent methods and their usefulness for each of the knowledge acquisition stages [88] . Some of the methods support more than one activity, but currently no method supports the entire knowledge acquisition process.

When we are looking for concrete guidelines on techniques to be used, the classification of Wei and Salvendy on cognitive task analysis (CTA) methods offers the most practical guidelines [147]. They define CTA as "the description of the cognitive skills needed to perform a task proficiently" ([147], p.6). It focuses on knowledge elicitation of decision making and other cognitive aspects of expertise [66]. Wei uses the mechanism-based classification, in which they identify four groups of techniques: informal methods (observations and interviews), process tracing methods (e.g., protocol analysis), conceptual methods (e.g., diagramming and error analysis) and formal methods (e.g., object-based model). Their work shows an overview of about 40 CTA techniques, a short description and (dis-)advantages of each technique. They make 11 recommendations on when to use each group of techniques, such as "When tasks

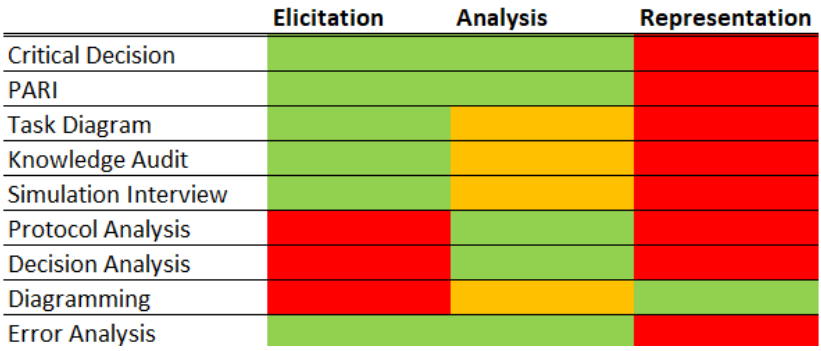


Figure 2.2: Classification of KA techniques according to the KA stage (adapted from Leu (2016)

or jobs do not have a defined domain, especially in the initial phase of CTA where we need to define and circumscribe the domain of tasks or jobs, select CTA methods in Family 1." [147]. Although this overview is very useful, it is too limited in at least two ways: i) it follows the classic viewpoint of knowledge acquisition as a one-off activity, while KBSs need continuous updating; ii) it requires a pre-analysis of the knowledge domain because the guidelines do not link a specific project type to the characteristics of the underlying knowledge.

None of the classifications elaborate on the role of the expert in the knowledge acquisition process. At the emergence of knowledge acquisition as a separate field, experts were mainly seen as "barrels" of knowledge, that could be "tapped" by knowledge engineers [88]. The experts played a passive role in the process, by which they were seemingly unaffected. Soon after, a transactional view on knowledge acquisition was advanced, as it became apparent that the transmission of knowledge is an interactive process, in which the expert plays an active role [88]. Knowledge engineers may ask questions that the expert cannot answer. By searching the answer, the knowledge of both the knowledge engineer and the domain expert grows [66]. This is also called the co-creation of knowledge. Other authors dream of a knowledge acquisition process without any knowledge engineer's involvement at all [104]. Their motivation can be of methodological nature, or may be efficiency based: as the domain expert has all the knowledge, why bother to bring someone else in? Some tools were developed to support domain experts in their efforts [52, 76], but this has not lead to a wide spread of the approach.

In the following paragraphs we discuss a selection of knowledge elicitation methods relevant for our case studies.

Interviews

Interviewing of domain experts by knowledge engineers is one of the most applied knowledge elicitation methods [28]. Generally speaking, two kinds of interviews can be discerned, i.e., structured and unstructured interviews, although other subdivisions such as direct/indirect interviews and explicit or implied questions are also described. It is considered as an informal method in the mechanism-based classification [28]. The big advantage of interviews is that the knowledge engineer does not need to be familiar with the domain. It is also an excellent way to establish a bond between the knowledge engineer and the domain expert [66]. On the flip side, the interviewer should have excellent interpersonal skills to guide the conversation, and analytical skills to extract the relevant information from the overload of unstructured information that is usually obtained in unstructured interviews. Unstructured interviews usually also take up relatively much time. Although the unstructured interview might be useful at the start of a knowledge acquisition process, to get a comprehensive overview of the domain it is expedient to turn to structured interviewing [88]. One example of a structured interviewing technique is the Task Diagram Approach to create an overview of tasks [101]. In the original view, task diagrams only provide a preliminary view that can be used for more in-depth interviews afterwards. Militello also executed a comparative study of three techniques (i.e., task diagram, knowledge audit and simulation interview) in two distinct case studies (one regarding fire fighting, and the other one regarding electronic warfare). Especially for the fire fighting case, the task diagram scored much better than the other two techniques on ease of use, flexibility, clarity of output and usefulness of information. For the other case the results for the flexibility of the technique and clarity of output were less positive compared to the simulation interview, although the overall result is still better than the results for knowledge audit [101]. Cooke [28] also mentions group interviewing techniques, as groups would produce more and better information than individuals.

Diagramming

Diagramming can be seen as a specific kind of graphical elicitation that can be used to stimulate people to remember additional information. Most research on graphical elicitation focuses on the use of photographs to stimulate the research object to think further than the obvious, and research on graphic elicitation by the use of diagrams is rather scarce. In his 2006 article on research diagrams as graphic elicitation stimuli, Crilly focuses on the unique characteristics, specific benefits and practical implications of using diagrams [29]. Diagrams differ from photographs in the fact that they are created by humans, rather than showing

the real world. Graphic elicitation with the help of diagrams is the result of three stages: the graphic ideation phase in which an idea is visualised, is followed by the graphic communication phase, in which the diagram is used to convey a message. In the third phase, the diagrams are used to solicit feedback. An overview of these phases are shown in Figure 2.3. Diagrams can be created by the researcher or knowledge engineer. Alternatively, they can be created by domain experts themselves. The difference is that the preliminary phases of graphic ideation and graphic communication are carried out by the researcher in the first case, whereas this is done by the domain experts in the second phase.

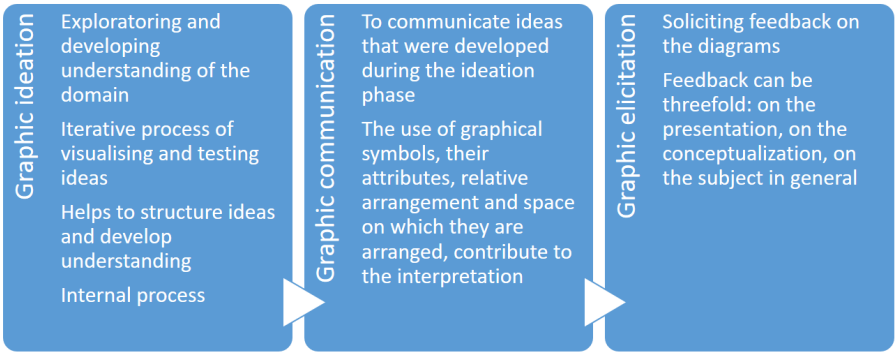


Figure 2.3: Three activities of graphic elicitation by means of diagrams [29].

The use of diagrams has some specific advantages. To start with, to create a meaningful diagram, the drawer needs to develop a coherent understanding of the domain. The act of diagramming itself stimulates the development of insight in the domain. In line with the idea that a picture is worth a thousand words, diagrams can also function as a communication language when parties do not share the same language, or ideas are difficult to verbalize. A diagram can delineate the domain of discussion, and prompt thoughts about parts that might otherwise be forgotten. Diagrams with different abstraction levels might be created, which allows to view the subject matter from different points of view.

There are also some factors to take into account when using diagramming. To start with, diagrams seem to be considered as starting points that are primarily useful to get an overview of the domain, but that should be completed with in-depth interviews [29]. Second, diagrams can be interpreted differently by different persons [12]. Third, different diagrams that highlight different aspects, can be used. Finally, some people prefer a textual representation of information over diagrams.

Prototyping

The use of prototypes is widely spread in the field of requirements elicitation for software development. The advantages that are typically associated with prototyping are the lower (development and requirements elicitation) costs, increased participation of stakeholders, relevant feedback that facilitates design decisions, and the ability to deliver a tangible result early in the project [51, 75]. Potentially, these elements contribute to better user acceptance and smoother change management. Typically a distinction is made between evolutionary prototypes, and throw-away prototypes. As implied by their name, evolutionary prototypes are constructed with the aim to evolve. The users already have a fairly good idea of what they need. By producing high-quality code in short sprints, functionalities are added to the prototype step-by-step [51]. Throw-away prototyping on the other hand, produces prototypes that are created really quickly with the idea of gathering information from the users. Users might not have a good idea of what they need prior to seeing and testing the prototype. The quality of the code is low [51]. One of the dangers is that quick-and-dirty code developed for a throwaway prototype, ends up in a production system. Another risk associated with the use of prototypes, is the lack of decent documentation. Moreover, evolutionary prototypes may take long to develop, and hence miss their target of a rapid elicitation tool. Also the initial enthusiasm of a tangible result early in the project, might backfire if management or users did not realize that the incorporation of a functionality in the production system can still take considerable time [51, 75].

2.2.2 Decision model and notation

The previous paragraphs focus on different elicitation techniques. The modelling of the knowledge obtained by it, is another important activity of knowledge acquisition. Also here, different techniques exist. According to their purpose, they can be assigned to one of the following categories ([44] in [71]):

- A phenomenon model is intended to be used by domain experts.
- An information model is developed for use by IT staff.
- A computer model can be used by the computer.

Hoffman [66] discerns models of the work domain, models of practitioner knowledge and models of practitioner reasoning. Although the author does not establish the link, these three models show a strong link with the three kinds of knowledge (domain knowledge, inference knowledge and task knowledge

respectively). In this thesis we focus on the explicit modelling of domain knowledge. Inference and task knowledge are immediately implemented in prototypes, whether or not supported by written procedures. In Section 2.1 we described FO(·) as modelling language. In this section we introduce DMN, another modelling language that we used frequently.

The first version of the Decision Model and Notation (DMN) standard was published in 2015 by the Object Management Group, that also published the Business Process and Model Notation (BPMN) and Unified Modelling Language (UML) [60]. The goal of the standard is "to provide a common notation that is **readily understandable by all business users**, from the business analysts needing to create initial decision requirements and then more detailed decision models, to the technical developers responsible for automating the decisions in processes, and finally, to the business people who will manage and monitor those decisions." ([60] p.15) The simplicity comes forth from the clear visual elements of the notation: both the structure of a decision and its logic are represented using unambiguous diagrams with only a few possible variants. As such, it can be considered as a diagramming technique. One of the largest advantages attributed to DMN is its usefulness as common communication tool when people do not speak the same (business) language.

Another important advantage is that DMN is a standard. This has a few interesting implications. To start with, we do not need to invent anything ourselves. The standard describes how information should be represented, and this representation is the result of a lot of experience in different cases.[4] Using a standard also means that the decision modelling skill is transferable from one situation/company/project to another and that external consultants and training are available. Finally, multiple tools are available for the notation and implementation.

To model a decision, the notation makes a distinction between the structure of the end decision, that is composed of sub-decisions, and the actual decision logic of the sub-decisions. The former is modelled in Decision Requirement Diagrams (DRDs), while the latter can be modelled in decision tables.

Decision requirements diagram

The decision requirements level models the dependencies between different (sub)decisions, their required inputs and outputs. DMN provides a visual notation for this in the form of DRDs. The number of different elements in a DRD are very limited: a rectangle that represents a decision, an oval that

[4]There is a vibrant decision management community, that challenges the limits of the standard, and proposes extensions and improvements to it. See https://dmcommunity.org/

represents an input, and a rectangle with clipped corners that represents a knowledge source. These figures are connected with the help of arrows. The input of a decision can either be an external input, or the output of another decision. In this way, a connected structure of sub-decisions can be created, as shown in Figure 2.4. In DMN, the decision structure is strictly hierarchical and non-cyclic, i.e., a "lower" decision cannot receive an input from a "higher" decision.

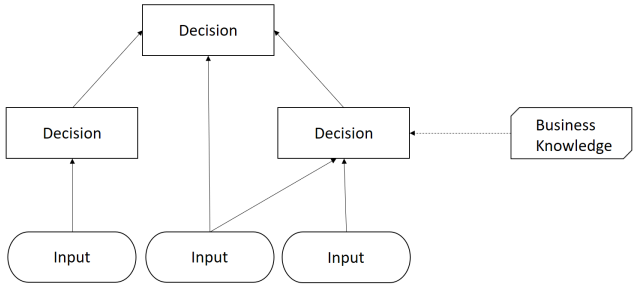


Figure 2.4: Building blocks of a DRD.

Using the DRD

Visualising the decision requirements in a DRD has several advantages. To start with, because the top decision is decomposed into lower level sub-decisions, the creation of the DRD assists the modeller to analyse the hierarchal structure of the domain. As most complex systems have a hierarchical or semi-hierarchical structure, this will usually be helpful [123]. It can also be used to structure and guide workshops or interviews to get the relevant information in an efficient way. The DRD is also useful to delineate the scope of the application, and focus the discussion. The finished model gives an overview of the external inputs that are needed. If relations, inputs, or intermediate decisions are incomplete (e.g., a decision box without arrows), this will show in the DRD and will prompt the knowledge engineer to ask further questions to create a consistent model. The DRD cannot be "implemented" or "executed": it gives an overview of the executable pieces of the decision logic, but it is not used for reasoning.

Decision tables

The second level identified by the DMN standard, is the decision logic level. The decision logic reflects how the inputs relate to the output. Although there are multiple ways to describe this relation, DMN favours the use of

standardised decision tables. In order to be executable, the DMN standard developed the Friendly Enough Expression Language (FEEL), and the more limited Simple Friendly Enough Expression Language (S-FEEL) that define executable semantics of the decision tables. An example of a decision table with its elements are shown in figure 2.5.

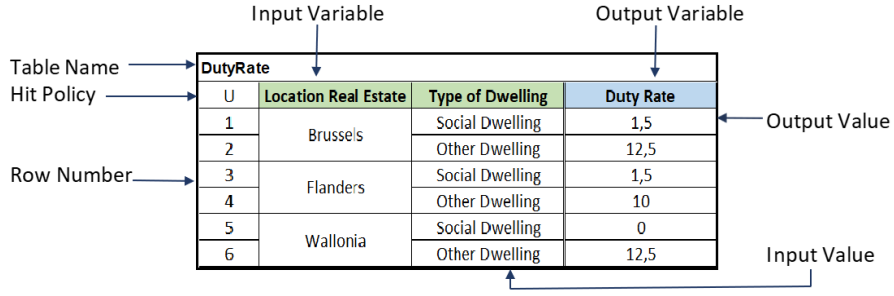


Figure 2.5: Example of a Decision Table with its elements.

Every table has a *hit policy* that is notated with one letter in the upper left corner under the name of the table. The hit policy is of major importance, as it indicates how the table should be interpreted. DMN features two kinds of hit policies: single hit policies in which only one output can be selected, and multiple hit policies, in which multiple outputs can be selected, possibly aggregated in some way. If there are no overlapping rules in a table, the interpretation is simple, as there is only one output for every possible combination of inputs. This *unique* hit policy is the default hit policy in DMN, and is strongly recommended. Other single hit policies are:

- Any hit policy (A) : two rows may overlap, but only if they have the same output.
- Priority hit policy (P) : rows may overlap, the indicated priority determines which output is selected.
- First hit policy (F) : rows may overlap, the applicable row highest up in the table is selected.

In all hit policies except the first hit, the order of rows and columns is irrelevant for the interpretation of the table, meaning that rows are independent of each other. This means that they can be ordered such that the resulting table is best readable.

For the first hit policy, the position of the rows determines which row is selected in case of overlap. Because this would lead to difficulties with the manual

validation and maintainability of the table, its use is strongly discouraged in the DMN standard itself, academic literature and practitioner's guidelines [60, 9, 20]. However, one might prefer to use first hit tables in two cases: i) If the use of first hit tables is firmly connected with business reasoning. This can either be because there are some clear and hard criteria that should be followed top-down. Or it can be because the business uses a strong default logic, as is the case in legal reasoning. In these situations the logic represented in first hit tables might be better readable than the logic represented in unique hit tables. ii) If there is a strong company specific reason that would make the use of another table format infeasible. There exist tools to convert first hit tables to unique hit tables [9]. The transformation of unique hit to first hit tables is straightforward. Therefore, both kinds of tables can be used next to each other in the same project during the different phases of the lifecycle of a decision table [92]. For example, during the knowledge elicitation phase rules can be elicited using a first hit table, that can be converted into a unique hit table for validation and verification.

The semantics of decision tables

Calvanese et al. [18] first translated the executable semantics of decision tables in S-FEEL in first order logic semantics. We shortly recall this for a unique hit table. Each column in a decision table corresponds to an FO constant, that is mentioned in the column's header. The S-FEEL data types for the header constants are number, string, boolean, days/time duration, month/year duration, time and date. Each cell in the column represents a condition that this constant may satisfy. An input condition Q is transformed into an FO formula $\Phi_Q(x)$ in one free variable x. The easiest case is when the condition consists of a term T (a term is either a constant of an n-ary function applied to n terms), which is short for the equality "$= T$". In this case, $\Phi_Q(x)$ is the formula $x = T$. If Q is an interval $[i, j]$, then $\Phi_Q(x)$ is $x \geq i \wedge x \leq j$. If Q is a list $(C_1, \ldots, C_n)$ of constants, then Φ_Q is $x = C_1 \vee \ldots \vee x = C_n$. Other kinds of intervals and comparisons are defined in a similar way. Table 2.1 summarizes a number of the possibilities.

Q	–	T	$not(T)$	$\leq T$	$(i, j]$	Q_1, Q_2
$\Phi_Q(x)$	true	$x = T$	$\neg\Phi_T$	$x \leq T$	$x > i \wedge x \leq j$	$x = \Phi_{Q_1} \vee x = \Phi_{Q_2}$

Table 2.1: The FO translation Φ_Q of different S-FEEL conditions Q.

For output values, the standard defines that each output entry shall be an arithmetic expression (such as an addition) or a simple value (numeric, string, boolean or date time literal). In other words, the conditions for output values

are more limited than the possible conditions for input values. It usually takes the form of a boolean or assignment (the first two cases in Table 2.1. We will call the output conditions QO

A row in a decision table connects the conjunction of input conditions with output conditions by implication. Each cell of a decision table (i,j) corresponds to a formula $\Phi^{ij}_{Q(x)}$ in one free variable x. A decision table with rows R, input columns I and output columns O is a conjunction of material implications:

$$\bigwedge_{i \in R} \left(\bigwedge_{j \in I} \Phi^{ij}_{Q(H_j)} \Rightarrow \bigwedge_{k \in O} \Phi^{ik}_{QO(H_k)} \right) \tag{2.1}$$

where H_j is the header of column j. For example, the first row of table 2.5 reads:

$$LocationRealEstate = Brussels \wedge TypeofDwelling = SocialDwelling \Rightarrow DutyRate = 1.5.$$

In the newest version of the DMN standard it is explicitly defined what should happen in case of incomplete tables: if no rules match, then the default output value is used, if there is no default output specified, then the output is *null*:

$$\bigwedge_{i \in R} \left(\bigwedge_{j \in I} \Phi^{ij}_{Q(H_j)} \Rightarrow \bigwedge_{k \in O} \Phi^{ik}_{QO(H_k)} \right) \bigwedge \left(\neg \bigvee_{i \in R} \bigwedge_{j \in I} \Phi^{ij}_{QO(H_j)} \Rightarrow \bigwedge_{k \in O} H_k = default \right). \tag{2.2}$$

In which $default$ corresponds to the user-set default value, or to *null* if no default value is set.

Creating decision tables

To build a table, one can start from the available information that can be read from the inputs in the DRD. Occasionally, it may turn out that a part of the input is missing, so the DRD needs to be complemented accordingly. Once all possible combinations of inputs are listed, the associated output can be determined subsequently. Another way of working is to start with the default case, and steadily expand the table with different exceptions, until all possibilities are covered. Using DMN tables to build decision logic, helps to get the relevant information in an effective way, hence leaving more time to focus on conflicting points of discussion. By starting from the possible inputs, the

completeness and consistency of the table can easily be checked and clarified. For larger tables, tools are available to support verification [9, 62, 128, 138]. It may occur that one does not always dispose of all information to complete a decision table. Sometimes, the input-rules are well defined, but the output is not. With the user-friendly setup of decision tables, the completion of the output can easily be assigned as homework for one of the experts.

2.3 Developing case study research propositions

In the previous sections we have described the state-of-the-art in the fields of KRR and knowledge acquisition. Knowledge acquisition is seen as an activity that encompasses three steps: elicitation, analysis and modelling. When going through the literature, some observations are noteworthy:

- The IDP system is a true KBS, and should have the flexibility to execute multiple kinds of calculations on the same KB.
- FO(·) as FOL-based modelling language is very expressive, and should be able to describe most domains.
- Knowledge acquisition is mainly seen as a one-off activity [66], while knowledge quickly evolves and outdated knowledge jeopardizes the utility of the system. Maintenance of the KB should be incorporated as an essential part of KBS development.
- The role of the expert in knowledge acquisition has shifted from "a barrel of knowledge" to an active participant in the interactive knowledge acquisition process, and even co-creation. Attempts to have experts develop KBSs independently, did not seem to get a foothold. As digitization progresses, domain experts should not only hold responsibility for their direct decisions, but also for automated decisions that build on their knowledge. Ergo, domain experts should be empowered to take responsibility over KBs and fulfil a pivotal role in knowledge acquisition. Nowhere have we found literature on how digitization changes the role of domain experts and what their responsibility towards automated decisions should be.
- Within the knowledge acquisition field, a multitude of methods and models exist. The idea that a combination of methods can be used, enters the academic literature [66]. A single tool or integrated toolset to cover the entire knowledge acquisition process, is currently missing.

- The knowledge acquisition process is currently seen as completely independent of the way the final models will be used in a system. Nevertheless, if a company decides to use an evolutionary prototype for knowledge elicitation, this schism cannot be maintained.

In the next chapter we propose some new tools and methods that we developed in the course of the case study research, and that come forth from these observations.

Chapter 3

Advancing the state-of-the-art

The previous chapter discussed the state-of-the-art in knowledge acquisition, modelling languages and KBSs. In using these techniques in real-life case studies, we ran into some limitations. This chapter describes our developments to overcome these limitations. The first section describes a new knowledge acquisition method, or rather, knowledge articulation method. This method is the first to span the three knowledge acquisition phases. In the next section, we describe the Interactive Consultant, a generic user interface that can be used as starting point to create evolutionary prototypes. The third section is devoted to cDMN, an extension of the traditional DMN standard.

3.1 Knowledge articulation

We concluded the previous chapter with the observations that i) in the current knowledge acquisition field there exists no end-to-end method, ii) maintenance is only an after-thought, and iii) knowledge engineers still take a prominent role, although the experts should bear responsibility for the KB. In this section we present our new knowledge articulation method that aims to remediate these gaps. We presented this method in the context of a product configurator development at the 10th World Mass Customisation and Personalisation Conference [8].

The main idea of this method is that domain experts should keep ownership

of the final decisions, even when parts of it are automated. Therefore, the domain expert should bear responsibility for the KB that underpins the KBS, and should be able to maintain it. To do so, the KB needs to be written in a language that can easily be understood and used by domain experts. Where knowledge acquisition is typically seen as the process of eliciting, analysing and modelling knowledge from a domain expert by a knowledge engineer [88], this now shifts to a holistic approach in which domain experts themselves express their knowledge in an executable model. Hence, the term *knowledge acquisition* does not cover the activity anymore; it rather becomes *knowledge articulation* performed by the domain expert, possibly with the support of a knowledge engineer.

The distinguishing feature of our method is the use of a single modelling language, resulting in a single model. This language should be simple enough to be understood throughout a company, by domain experts and IT staff. Yet, it should be sophisticated enough to capture the complexity of the domain. The model should be executable, such that the commonly understood model is the unique source of information for the KBS. Using a single model decreases the cost and risks associated with the creation of multiple models. With the use of a simple modelling language, domain experts can create the models by themselves, or in collaboration with the knowledge engineer. The creation of the models is no longer an activity performed by the configuration engineer to acquire knowledge from domain experts, but rather a proactive articulation of this knowledge by the domain expert. The difference between knowledge elicitation, analysis and modelling hence disappears in one holistic model creation activity, driven by the domain expert.

The second central element of our method is the focus of the modelling activities on the decision logic instead of product details. This makes the process of collecting information easier, as domain experts can go step by step through the decision process they use day by day instead of focusing on all possible abstract cases. Moreover, the user of the model can immediately trace back in which case which value is used.

The use of rapidly evolving prototypes is the third pillar of this method. The use of prototypes is widespread in the field of requirements elicitation for software development, but can also be helpful to validate the content of the KB.

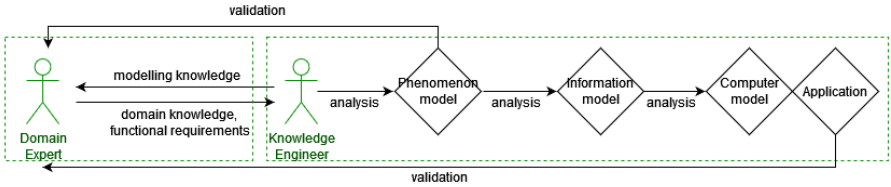
Finally, if the company has multiple experts on the domain, we recommend the workshop format to create the decision models together. Involving multiple experts has some advantages for the knowledge acquisition process and its results. Some cited advantages are the speed of the process (compared to a back-and-forth discussions over mail for example) and a result that would be more concise [47]. When a graphical modelling language is used, the workshop format allows

the participants to go through the ideation phase together. Moreover, according to the unified theory of acceptance and use of technology, the active involvement of experts stimulates acceptance of the application, either directly (because of the involvement), or indirectly (because of the expert's social influence) [131].

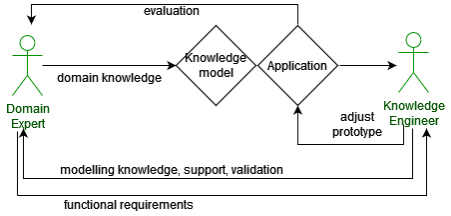
Proposition : Domain experts should have a central role in creating , using and maintaining a KBS		
Characteristics	**Impact**	**Implementation**
- Use of a single model by domain experts, IT experts, and computers - Modelling of decision logic (rather than products or processes) - Use of rapid evolutionary prototyping - Multiple experts workshop format when possible	- Experts maintain ownership of the KB and decisions that are automated by it - Maintenance of the KB can be integrated in the regular work routine - Improved expert acceptance of the KBS	- Use of cDMN and DMN as modelling language - Straight-through translation of decision model to FO(.) specification - Use of Interactive Consultant as prototype

Figure 3.1: Summary of the knowledge articulation method.

Table 3.1 summarises the characteristics of the method. Knowledge articulation is a completely different way of formalising knowledge compared to the traditional knowledge acquisition methods. Remember that the knowledge acquisition process is traditionally seen to consist of (at least) three stages: knowledge elicitation, analysis and modelling. In Figure 3.2a this is illustrated. In the knowledge elicitation step, knowledge from the expert is made available to the knowledge engineer. The knowledge engineer analyses the information, and creates a phenomenon model. This phenomenon model is sent back to the domain expert for validation. However, in order to validate the model, the domain expert needs to be able to interpret the model. Therefore it might be necessary that the knowledge engineer educates the domain expert in the modelling formalism. If the phenomenon model is validated, the knowledge engineer will engage in analysis and modelling to create the information model. After a third round of analysis and modelling, the computer model that forms the application, is obtained. The application then needs to be validated by the domain expert. If something needs to be changed, either because it is not implemented properly or because the environment changes, the change needs to go through the entire process again. This process stands in contrast with the knowledge articulation process demonstrated in Figure 3.2b. In this process, domain knowledge results immediately in a unique model, that is used as knowledge model in a prototype application. The knowledge engineer plays a supporting in role in this. After testing the prototype, new requirements can be passed on to the knowledge engineer, who tweaks the system to the user's needs. Changes in the domain can be updated directly by the domain expert.



(a) Knowledge acquisition process



(b) Knowledge articulation process

Figure 3.2: Illustration of kinds of knowledge in different application types

3.2 Interactive consultant prototype

In the knowledge articulation method we use a prototype to solicit expert's requirements. Prototypes are not only useful as part of the knowledge articulation method, but also in combination with other methods. A generic user interface that can be used with the IDP system and FO(·) KB, called *Interactive Consultant* [1], was developed. Its generic, domain independent use, is the result of the strict separation of knowledge base and inferences. On the IDP homepage[1] several interfaces are available for try-out. These interfaces can be used on any problem domain by simply replacing the underlying knowledge base by one's own knowledge base. Hence, these user interfaces can be used to quickly develop new prototypes.

The Interactive Consultant can be considered an evolutionary prototype in two senses. The underlying KB consists of modular formulas. A first prototype may hence contain only a fraction of its finalized version. Formulas can be added step-by-step to reach this. Also, the inference tasks that are included are independent of one another and of the KB. New inference tasks can be developed and integrated as preferred by the users. Yet, although the prototype can be considered evolutionary, it can still be developed quickly. We also notice

[1] https://krr.gitlab.io/demos/

that some companies prefer to use a layout scheme specific to their brand. They use the prototype as a throw-away version.

The Interactive Consultant interface includes the most important inference tasks.[2] Figure 3.3 contains a screenshot of the interface with the explanation inference.

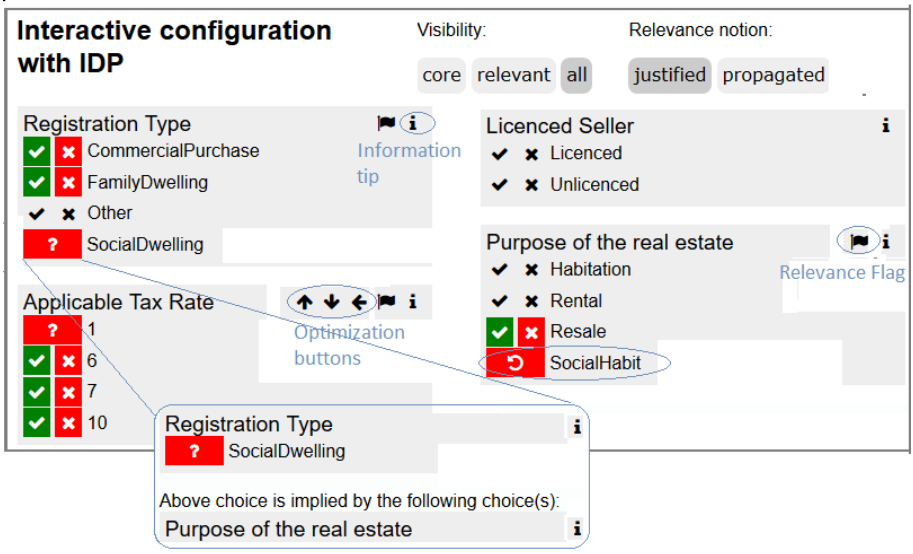


Figure 3.3: Interactive Consultant with explanation inference.

Apart of the model expansion, propagation and optimisation inference tasks mentioned in 2.1.5 the inference tasks in the interactive consultant are:

- Abstract Model Generation: this shows an abstract representation of the possible models in tabular format.
- Relevance: In IDP, no distinction is made between input and output variables. However, a set of variables can be assigned as “core”, meaning that these are the variables one is interested in. If the core is determined, no other undetermined values are relevant anymore, because the value of the core has already been determined. The relevance inference highlights the relevant variables that need to be determined to get a value for the core variables.

[2] https://interactive-consultant.idp-z3.be/

- Explanation: The explanation inference identifies the chosen atoms that imply a propagated atom, and allows to revise these choices. In the interface this means that on can click on a propagated atom (identified by the "?"), and a pop-up window shows the chosen atoms that cause the propagated value [3].

The Interactive Consultant is a good start for a prototype, and has been developed further for specific use cases. My role in the creation of the prototype was suggesting new functionalities based on my case work. Jo Devriendt developed the first version with the current visuals, and that we used extensively in the development of the legal application (Chapter 4). It was based on the Autoconfig interface on the IDP homepage that was developed by Ingmar Dasseville [30]. The current online version that uses IDP-Z3 is being developed by Pierre Carbonnelle, who also supports further development together with Simon Vandevelde, Jo Devriendt, Bram Aerts and Matthias van der Hallen. The original idea for this interface stems from Hanne Vlaeminck, Joost Vennekens and Marc Denecker [143].

3.3 cDMN : constraint decision model and notation

The DMN standard that we introduced in the previous chapter has considerable advantages with regards to interpretability and ease-of-use of the decision tables. However, the standard also has a considerable disadvantage, namely its limitation to rule based decisions. A decision table assigns a specific value to the output variable based on the combination of input values. If the table is incomplete (i.e., a combination of input factors is not defined), the output linked to it is the default value or *null*. Each input row is a sufficient condition for the associated output value. Together, the disjunction of input rows that lead to the same output value, are a necessary condition for this output value.

It is not possible to exclude values for the output variable (i.e., to limit the solution space), as is the case for constraint based approach. In this section, we introduce constraint Decision Model and Notation (cDMN), that aims to expand the tabular notation of DMN with a similar constraint notation. In many real-life situations, a decision is steered by constraints. For example: the buyer of cigarettes needs to be at least 18 years old. This regulation does not enforce a specific age on the buyer of cigarettes, but it does constrain the group of potential buyers to all adults.

To model constraints, we developed cDMN, a constraint notation that can be used alongside and together with the DMN standard. This section describes work done together with Bram Aerts and Joost Vennekens as reported in [1]. cDMN was further developed and implementation tools were developed by Simon Vandevelde, Bram Aerts and Joost Vennekens [137].[3]

3.3.1 Constraint tables

Where a decision table is the core element to represent decision logic, a constraint table is the core element to represent constraints. A DMN decision table uniquely defines the value of its outputs. This is due to two properties. First, while cells in an input column may contain different kinds of S-FEEL expression, only single values can be used in output columns. Therefore, if a row matches the input, the corresponding single value must be assigned to the output. Second, DMN specifies an output if no row matches the input. The output is the default value set by the modeller. Alternatively, when there is no default, the output is assigned the special value *null*, which is typically taken to indicate an error in the specification.

Our new constraint tables change both of these properties. First, no default or null values are assigned in constraint tables. This means that if none of the rows are applicable, the outputs can take on an arbitrary value, as opposed to being forced to a default value or *null*.

Second, we allow the same S-FEEL expressions that can be used in input columns to appear in output columns as well. This can be demonstrated with a small example taken from the case in which a KBS is designed to assist with the configuration of a product, for which we need to select a material. Moreover, we enhance the expressivity of constraint tables by allowing the use of types and relations in the column headers instead of simple constants. The *Material of Body* table in Figure 3.4 states that if a closed design is used, material M2 cannot be used, without specifying which of the other materials should be used in its place. We can now model universal quantification by using a type as input header, and a dash in the input field. The *Operating Condition* table in Figure 3.4 states that for every material that is used holds that the temperature to which it resists, should be higher than the operating temperature.

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 one of these implications must be satisfied. In the same example of product configuration, a design needs to be selected. The Design Type table in Figure 3.4 states that

[3] www.cdmn.be

Material of Body		
E*	Design Type	Material of Body
1	Closed	Not(M2)

Design Type		
E*	Pressure	Design Type
1	>150	Not(Closed)

Operating Condition		
E*	Material	Temperature Resistance of Material
1	–	> Operating Temperature

Figure 3.4: Constraint tables for body material and design type.

if the pressure exceeds 150, the design type must not be Closed. The effect of its E* hit policy is seen when the pressure does not exceed 150. In this case, our constraint table imposes no restriction on the design: an open and a closed design are both possible. The Material of Body table combines the E* hit policy with the ability to use S-FEEL expressions in the output. It states that if the design is closed, material $M2$ cannot be used for the body. Again, if an open design is used, no constraints are imposed on the body material. We can now model universal quantification by using a type as input header. The *Operating Condition* table in Figure 3.4 states that for every material that is used, it holds that the temperature to which it resists, should be higher than the operating temperature.

The semantics of constraint tables

In Section 2.2.2 we repeated the formalisation of the semantics of DMN tables. Here we expand the formalisation to constraint tables.

In constraint tables, we represent each condition Q in a table row by the same formula Φ_Q, as shown in Table 2.1. Φ_Q now represents both input and output conditions. Moreover, we also expand the possible data types in the column headers so they can represent types and n-ary relations besides the traditional S-FEEL datatypes (number, string, boolean, days/time duration, month/year duration, time and date). In a constraint table we do not force a default value in case the table in incomplete.

For instance, the first row of the "Material of Body" table corresponds to the formulas $DesignType = Closed \Rightarrow MaterialOfBody \neq M2$. The semantics of constraint table T is then defined as the conjunction $\bigwedge_{r \in R} \Phi^{\Rightarrow}_{H}(r)$.

Limitations to cDMN

We discussed how cDMN allows a straightforward notation for universal quantification ("for all") by using a type in the header as in the "Operating Conditions" table. The current cDMN notation does not allow the use of existential quantification. The reasons for this are twofold. First, existantial quantified formulas can be reformulated to universally quantified formulas using DeMorgan's laws for equivalence or by using aggregates

$$\forall x : P(x) \equiv \neg\big(\exists x : \neg P(x)\big)$$

$$\exists x : P(x) \equiv \neg\big(\forall x : \neg P(x)\big)$$

$$(\exists x : P(x)) \;\Leftrightarrow\; count\{x : P(x)\} \;\geqslant\; 1.$$

To continue the example of product configuration, one of the constraints is that for each component there must exist a material, such that the used material can resist the temperature it needs to function in :

$$\forall\, c[Component] \;:\; \exists\, m[Material] \;:\; UsedMaterial(c, m) \wedge ResistTemperature(m).$$

This can also be reformulated as:

$$\forall\, c[Component] \;:\; count\{m[Material] : UsedMaterial(c, m) \wedge ResistTemperature(m)\} \;\geqslant\; 1.$$

Another example is the case in which registration duties to be paid are calculated. One of the constraints is that for each property, only one tax rate is applicable. This can either be formulated as:

$$\forall\, x[House]\; y[TaxRate]\; z[TaxRate] \;:\; AssignedTax(x, y) \wedge AssignedTax(x, z) \;\Rightarrow\; y = z.$$

For all two tax rates assigned to a house, they must be the same. Or:

$$\forall\, x[House] : \neg\exists\, y[TaxRate]\; z[TaxRate] \;:\; AssignedTax(x, y) \wedge AssignedTax(x, z) \;\wedge\; y \neq z.$$

There do not exist two tax rates such that both rates are assigned to the house.

Note that the implication that was used together with the universal quantification, is replaced by a conjunction in the sentence with the existential quantification. To model it in cDMN we would need a kind of constraint table without implication.

The second limitation of cDMN is that there exists only one hit policy for constraint tables for the moment, i.e., the *every* hit policy that states that all applicable constraints should be met. In our case studies we did not come across any situation for which another hit policy would be needed.

Third, in FO($\cdot$) constraints may exist of several nested implications and equivalences that cannot be elegantly represented in a constraint table. The description of the inner working of an OR-gate is an example of this: $\forall\ x : And(x) \Rightarrow (On(FirstInput(x)) \wedge On(SecondInput(x)) \equiv On(Output(x)))$.

3.3.2 cDMN and other formalisms

We developed cDMN with the purpose of expanding the user-friendly tabular notation to constraints. It should be noted that also within DMN the possibility exists to model some constraints by using some of the more advanced modelling possibilities, such as boxed notations, iterations and functions. However, then the appealing visual representation of simple decision tables gets lost. We have described the semantics of a constraint table by translation to FOL. In theory it is possible to express the semantic in other formalisms. The only requirement is that the formal language is expressive enough to express the complexity of a constraint table. For example, if we look at DLs that we described in Chapter 2, we expect that they will not be sufficiently expressive because DLs are less expressive fragments of FOL, and cDMN is based on a more expressive extension of FOL. Although it requires more thorough investigation, some differences appear at first sight. For example, in DL a predicate may have at most two arguments. This kind of limitation does not exist in cDMN. On the other hand, $\mathcal{AL}$ and $\mathcal{FL}$ allows limited existential quantification, and the extension $\mathcal{E}$ allows full existential quantification, which is currently not supported by cDMN.

3.4 Conclusion

In this chapter we described new developments in the field of knowledge acquisition for the creation of a KBS. They intend to fill the gap in the existing tools, methods and techniques that we stumbled on with the execution of our

case studies. In the Chapters 4, 6, 8 and 10 the cases and how they lead to the developments are described.

The construction of an encompassing knowledge base of domain knowledge remains a massive and resource-intensive mission. In using (c)DMN, the advantages of a diagramming technique are combined with the advantages of a user-friendly standard. cDMN is able to model real-life constraints, and complements DMN that can be used to model if-then rules. The (c)DMN KB can be automatically translated to FO($\cdot$). In combination with the interactive consultant user interface, this allows for the quick creation of an evolutionary prototype.

These tools can also be used in the knowledge articulation method. This method relieves some of the modelling burden and provides an efficient alternative for traditional multi-stage approaches because it integrates the knowledge acquisition, analysis and modelling phase in one tool-supported method. Moreover, it puts the role of the domain expert and their domain knowledge in the heart of the knowledge acquisition process.

Our method can be applied in different circumstances: with one or multiple experts, and in different domains, but following factors seem to impact the suitability of the method:

- Our method is easier to use if the experts consider the modelling language "easy". As we use (c)DMN for modelling, this means that it is easier to use by experts who are used to structured analytical thinking.
- Although the rapid prototyping is suitable to reveal both domain and functional requirements, there probably exist better tools when there are a large number of tacit preferences to discover.
- The modelling of the engineering process instead of product details probably means that our method is less suitable to build configurators for mass-customisation where clients have their individual decision process.

Chapter 4

Case: Calculation of registration duties

In this thesis we investigate the suitability of KBSs and approaches to create them by means of case study research. This chapter describes the case of calculating registration duties. It focusses on the followed knowledge elicitation, modelling and analysis approaches, and describes the resulting application prototype. The generalisation of lessons learned and embedding in legal logic literature is part of the next chapter (Chapter 5). After introducing the case, we describe how the different models were created, and how we adapted an existing user interface to meet the requirements of the notary. The case was conducted with the help of notary of office Van Pelt in Antwerp as legal expert. Our colleague Jo Devriendt developed the interactive interface and adapted it to the needs of this case. The content of this chapter is based on the work presented at RuleML2018 [5] and the IEEE Conference on Semantic Computing [3].

Contributions In this case we developed two DMN models and two FO($\cdot$) specifications for the legislation on the calculation of registration duties. This demonstrates the expressivity of the modelling languages to model the legislative domain. One set of models (DMN and FO($\cdot$)) reflects the legislation prior to the significant legislative change of May 2018. The other set of models reflects the new legislation. Because of these double models, we were able to demonstrate the maintainability of the FO($\cdot$) KBS. We developed an implementation of both models and demonstrated their weaknesses and advantages for use in this case.

We contributed to the development of the interactive consultant prototype by analysing the functional requirements and translating them into inference tasks.

4.1 Introduction to the case

In Belgium, when a party wants to conclude a transaction on the real estate market, a notary is required to affirm the process, providing legal certificates for the requested transaction. This registration gives rise to the payment of registration duties, which depend on the region in which the estate is situated. Duties may deviate from the standard rate depending on characteristics of the buyer, the property, the seller and the location of the real-estate. For the Flemish region the applicable legislation is the "Decreet van 13 december 2013 houdende de Vlaamse codex fiscaliteit" with amending decrees from December 19th 2014 and May 18th 2018.

Prior to June 1st 2018, most family houses would either be subject to the standard duty of 10%, or to a reduced duty of 5% for "modest houses". Whether a house would classify as modest depended mainly on its *kadastral income (KI)*, a value which represents its theoretical rental value. This KI was then compared to a threshold, that depended on the buyer's number of children. In addition, there also existed an independent and elaborate system of *abattements* (reductions on the taxable base of the house).

To remedy the complexity of this system, new and simplified legislation came into force in 2018. The concepts of KI and *abattement* were abandoned. To determine if a house should be considered "modest", its actual selling price is now used instead of the fictitious KI. These reforms of the registration duties represent a profound change: of the original 42 law articles of chapter 9 concerning the registration law, the decree of 18 May 2018 abolishes 4 articles, modifies 9 and adds 5.

In the case we create an application to support the notary during client interviews to gather all relevant information and decide on the most advantageous tax rate.

4.2 Requirements

At the start of the case study the application requirements were rather vague. The notary wishes an application that is able to instantaneously use information from the moment it becomes available. For example, if the user says that they

want to buy a house as a real estate broker with the aim to renovate and resell it at short notice, the next question should not be if the buyer will establish his domicile there. This comes on top of the minimum baseline requirements of the application:

- *Completeness and correctness* The application should ensure that all possible discounts are taken into account, and only rule out or apply discounts when warranted by the information provided by the notary.
- *Optimisation* Some of the tax exemptions are not cumulative, the implementation should allow for the optimisation of the amount to be paid.
- *Usability* In meetings between notaries and their clients, the technology should not disturb the confidential atmosphere. A lot of typing and searching for the correct button is out of the question.

After evaluating a prototype of the application, the notary office came up with these additional requirements:

- *Traceability* The decision outcomes calculated by the application should be easy to check and explain, in order to increase clients' confidence in the application.
- *Efficient information gathering* Only questions relevant to possible discounts should be asked. E.g., as soon as it is clear that one of the discounts cannot be used, questions related to this discount become irrelevant and should no longer be asked.

4.3 Decision model

The calculation of registration fees depends on multiple sub-decisions: the value of the property taking into account possible discounts, the applicable tax rate, and portability of previously paid registration duties. The use of the DRD as visual overview of the different sub-decisions and their dependencies is a convenient starting point. It helps to understand the problem at hand, and to structure the development of both the decision tables and the FO($\cdot$) model.

4.3.1 Followed approach.

For this case, multiple overview models with different levels of granularity were developed. At first a rather high-level DRD (upper right panel of Figure 4.1) was developed.

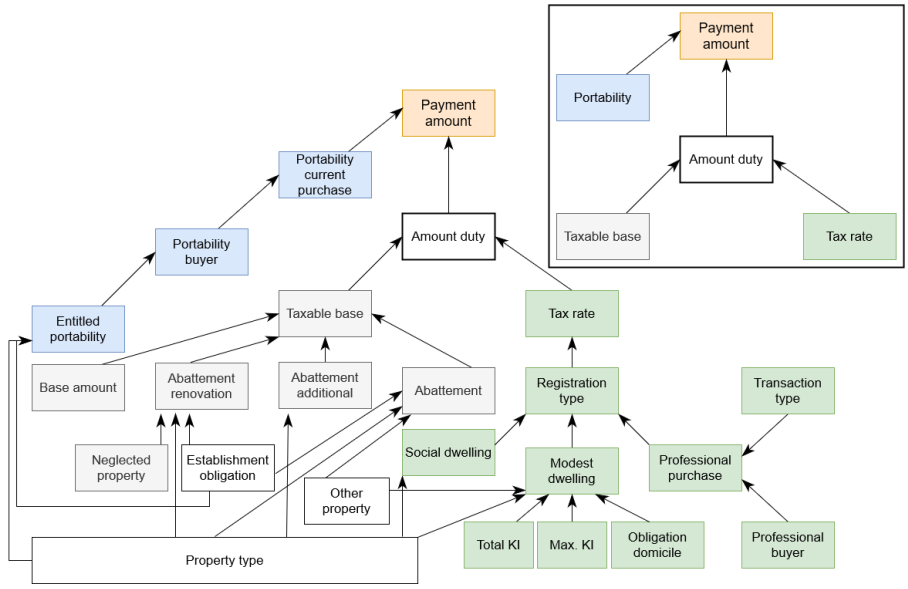


Figure 4.1: DRD of registration duty calculation with abstract (top right) and expanded view.

The advantage of this figure is its simplicity: for readers it is easy to understand which elements impact the registration duties. However, the abstraction level is too high to use it as a practical input for the decision tables. Therefore a second DRD with all decision nodes was developed (see left panel of Figure 4.1). Although this one is complete, it loses the advantage of comprehensibility of the high level model. Of course both DRDs are interrelated in the sense that the elaborate DRD details the decisions that were captured in one decision node in the high level DRD. It is an intuitive extension of the DMN standard to expand the abstract top right DRD in the bottom left finer-grained DRD.

Next, the decision tables were constructed. This can be done in different ways, depending on the hit policy of the tables. For single hit tables with the unique hit policy, rows must be mutual exclusive and it is considered a good design principle to represent all possible combinations [139]. The relevant legislation states a default tax rate (i.e., 10%) and identifies several discounts which are

each applicable in a specific situation that is characterised by a conjunction of multiple conditions. First hit tables are well suited to model this kind of default reasoning. In a first hit table, the default value is displayed at the bottom with a *catch remainder* rule. The modeller might start by identifying this rule. Then the exceptions are modelled in the rows above, in order of importance. The standard tax rate (i.e., the default value) is 10% in Flanders (section 4, subsection 1 art. 2.9.4.1.1: The sales duty is 10%). In subsection 2, deviations from this general rule are described [108]. Each of these exceptions has specific conditions that need to be fulfilled to be entitled to the abated duty. This kind of default reasoning is often found in legislation, and can be comprehensibly represented in first hit tables. The example in Figure 4.2 shows a unique hit and first hit table for the determination of the lower tax rate of 6%, applicable for family dwellings. In derogation from article 2.9.4.1.1, the registration duty amounts to 6% for agreements regarding *pure purchase*, whereby one or more *natural persons* acquire *full ownership* of a *house* in order to establish their *principal residence* there (art. 2.9.4.1.11). The corresponding complete unique hit table is shown in Fig. 4.2a. The table has a very characteristic triangular shape, because it needs to contain a row for every exception, to express a conclusion if it does not meet the required value. Such a representation can be confusing: it gives the impression that there are multiple possible cases when calculating the decrease tax rate, while there are in fact only two cases: the standard ten percent, or the decreased six percent. Alternatives are the use of first hit tables, as demonstrated in Figure 4.2b, or the use of in incomplete unique hit table with a default value. Although in this example unique hit tables with default value are even shorter than first hit tables because the catch-all final row is missing. The default value is underlined in the list of possible values. To know this, the reader needs to be familiar with the DMN standard. Although it should be confirmed by experiments, the interpretation of the default catch-all row, might be easier to guess without prior knowledge.

Following the generally accepted good practices, we initially decided to exclusively use unique hit tables. This resulted in a model consisting of 13 tables with a total of 102 rows. Because we noticed the typical triangularly filled shape in the U tables, we decided to revise the model and use first hit tables. This reduced the number of rows by more than 30% (70 rows instead of 102). This reduction of rows resulted in an increase of the readability of the model.

However, from the point of view of verification and validation this solution is considered bad practice. By validation we mean that the user checks if the table correctly assigns output values to the input conditions as meant by the user. It is different from but related to verification of a table, which is an (often) automated check to establish the correct formatting of a table according to the

Tax rate						
U	Pure Purchase	Buyer	Ownership	Type	Principal Residence	Tax rate
1	No	–	–	–	–	10
2	Yes	Not Natural	–	–	–	10
3	Yes	Natural	Partial	–	–	10
4	Yes	Natural	Full	Not House	–	10
5	Yes	Natural	Full	House	Yes	6

(a) Complete unique hit table to calculate decreased tax rate

Tax rate						
F	Pure Purchase	Buyer	Ownership	Type	Principal Residence	Tax rate
1	Yes	Natural	Full	House	Yes	6
2	–	–	–	–	–	10

(b) First hit table to calculate decreased tax rate

Figure 4.2: Comparison between U and F table for tax rate.

DMN standard [9, 62, 128]. For a complete unique hit table, verification results in a table with disjoint rules, that together cover all possibilities. A verified unique hit table, is more likely to also be valid, as typical mistakes such as gaps and overlaps, are already caught by the verification step. This is different for F tables: they typically contain a last "catch remainder" default row, which makes a completeness check superfluous. It also does not make any sense to check for overlapping rules, as overlaps are allowed. There is one modelling error unique to F tables: the so-called *unfireable rules*. These are rules that can never be applicable, because they are shadowed by the above rules (with higher priority). The verification of unfireable rules contributes to the user's validation of the table, but to a limited extent, as other important sources of mistakes such as unintended gaps and overlaps, are not identified.

By creating both a unique hit and first hit version of the same decision table, the advantages of both kinds can be combined. If the decision tables are originally constructed in a first hit policy, algorithms such as the one from Batoulis [9] can be used to transform them to U tables. The user is then able to use more powerful verification tools that verify the tables and may support him in validating the tables. It can be converted back in the first hit format afterwards. Maes and Van Dijk use the notion of "lifecycle of a decision table" to refer to the shift of focus during the creation, validation and use of decision tables [92]. It determines which kind of table is best suited for a specific action.

4.3.2 Result and implementation

OpenRules was used as decision engine as it is available for free and has a low entry effort because it is based on Microsoft Excel files. Although the development of a DRD is recommended as a first step, the program itself does not contain DRD-drawing functionalities. The DRD can be included in the executable file as documentation, but this is optional as it does not impact execution. The OpenRules model consists of 25 decision tables, grouped in five areas: determine portability, determine taxable amount, determine tax rate, calculate portability and calculate registration duties.

The explicit assignment of hit policies is not possible in OpenRules, but the execution of single hit tables finishes after a hit is found, i.e., de facto the first hit policy is used.

OpenRules evaluates tables in a context-dependent way. For instance, the KI has to be checked for all previously acquired properties of the buyers . This is modelled with a decision table that is ran several times with the use of an iterate table that defines the array of objects and the decision table that should be ran for these objects. For each property some information is introduced, e.g., if it is a plot or a house, and the value assigned to the real-estate, called KI. This information is also needed for the actual purchase itself. The same variable name can be used in both situations and the meaning is derived from the context.

OpenRules executes the specified rules in a predefined order. When a part of the input values is missing, the output value for this decision is forced to -1. Rules that can be executed with the available information will be executed, which may lead to a solution that simply ignores unknown facts.

The data modelled in OpenRules can be used to find a solution based on the available inputs. A distinction is made between input variables, internal variables (i.e., intermediate results), and output variables. The value of an internal variable cannot be set manually. Instead, values provided for internal or output variables will be checked against the calculated outcome, and deviations will be reported as an error. The format allows manual checks for correctness and completeness and proves to be useful in the process of knowledge elicitation. DMN tables in theory allow for reverse reasoning from output to input variables, but this functionality is not supported by OpenRules. The optimisation of registration rights is done by introducing different variables for the different options and comparing them in a later stage, as demonstrated in Figure 4.3.

DecisionTable RegistrationDutiesAbattement
Action
RegistrationDutiesAbattement
(PurchasePrice - Abattement - AdditionalAbattement - RenovationAbattement) * TaxRate/100

DecisionTable RegistrationDutiesPortability
Action
RegistrationDutiesPortability
((PurchasePrice - RenovationAbattement) * TaxRate/100) - TotalPortability

DecisionTable Advice		
Condition		Message
RegistrationDutiesAbattement		**Explanation**
>	RegistrationDutiesPortability	Use Portability
<		Use Abattement
=		Use Portability or Abattement

Figure 4.3: Optimisation in OpenRules.

4.3.3 Evaluation

The creation of a DRD is helpful to ensure a correct analysis of the structure of the law text. Moreover, it helps to redirect discussion with the notary to the scope of the application as depicted in the DRD. The decision tables were instrumental to make sure the complex juridical jargon was understood correctly, as they are easily understood by both the legal domain expert and the modelling knowledge engineer. The hierarchical decomposition of the domain in smaller decisions that are linked with each other by using the output of a table as input in a higher level decision, is essential for this. The requirement to formalise all variables in a glossary helps to keep overview.

The OpenRules implementation has some shortcomings that prevent it from being an acceptable tool for the notary. First, there is the behaviour of the application in case of missing data. Missing data is forced to -1, which results in ignoring the rules that use this information. There might still be a solution, based on the partially available information. Although this way of working might be desirable in some situations, it does not comply with the way the notary wants to use partial information. He prefers to use available information to narrow down the complete domain of possible solutions, keeping visible other options on which no information is currently available. Second, all input data needs to be inserted at once, leaving no room to interactively select relevant information. The implementation misses an important feature of interactivity. Third, the implementation is only able to calculate tax rates based on the input. It cannot reason backwards, i.e., given a tax rate, what could have been the input values? Although this functionality was not explicitly requested by the notary, it is very useful to make simulations or to detect unexpected results, which is possible using backwards reasoning. As the OpenRules implementation does not meet these needs, we look for a better solution that allows reasoning with partial information (instead of neglecting it), that is interactive, and allows

reasoning in different directions

Although the implementation in the OpenRules system did not meet the expectations of the notary, the DMN model served well as intermediary model. As the DMN notation is easy to read and understand, and is almost syntax-less, it is convenient to create such an intermediate model that focuses on understanding domain knowledge rather than moulding it into the required format of the system. In the specific case of legal knowledge, the twofold structure with DRDs that reflect the structure of the law texts, and decision tables that follow articles of law, is an additional advantage of DMN. The use of OpenRules allows to define concepts and attributes in a business glossary, while rules are formalized in decision tables. These parts are analogous to the *class hierarchy* and *rule base* parts suggested by [11]. Especially in [11], the importance of structuring the model in the way that reflects the structure of the legislation, what they refer to as *isomorphism*, is stressed. While our DMN model shows some isomorph characteristics, we sometimes deviate from the principle. For example, we create registration types that characterise the applied rate, e.g., social dwelling, professional purchase. In the model of our application, registration type is defined in one table, with each rule referring to a separate article. The tax rate is defined in a separate table. In the legislation does not exist a name for the different tax rates. There is no centralised overview of the rates either: they are found in the different articles. The dogmatic use of isomorphism seems less relevant in the limited scope of our application and with the implemented features of explanation and relevance.

4.4 FO(.) specification

As the functions of reasoning with partial information and optimisation of registration duties that are important for the notary's office are not performed satisfactorily by OpenRules, we created a prototype using the IDP system. From evaluation of the OpenRules implementation, it became apparent that it did not meet the notary's expectations. However, it was difficult to formulate exactly the functionalities that he did desire. To elicit these requirements, we transformed the DMN model to a FO(·) specification, and used it as KB for the online available GUI to create a prototype.

Dasseville et al. [30] demonstrate that DMN-tables can be translated to FO(·) definitions. The DMN tables that were constructed as starting point, are in this step translated to FO(·) definitions one by one. Similar to the format of first hit tables with default values the bottom, default values in FO(·) are formulated as selected if none of the other options are selected. For instance, the decision

table in Figure 4.2 can be formulated as:

$$\begin{aligned}\mathrm{TaxRate} = 6 \leftarrow\ & \mathrm{PurePurchase} = \mathrm{true} \wedge \mathrm{Buyer} = \mathrm{natural} \\ & \wedge\ \mathrm{Type} = \mathrm{house} \wedge \mathrm{Ownership} = \mathrm{full} \\ & \wedge \mathrm{PrincipalResidence} = \mathrm{true}.\\ \mathrm{TaxRate} = 10 \leftarrow\ & \mathrm{TaxRate} \neq 6.\end{aligned}$$

The translation of variables from OpenRules to IDP can happen in multiple ways. Consider for instance the KI of a specific property: it was captured as the variable Ki in OpenRules, and depending on the context this pointed to the Ki of the current purchase, or the KI of previously acquired real estate. In FO(·) this can be solved by creating a function that maps all properties to their KI, as shown in the right panel of Figure 4.4. This option seems logically consistent, but it could lead to the use of (less preferred) partial functions for attributes of the property that are relevant for the current purchase but not for the other real estate. A way to circumvent this, is to create a function specifically for previously acquired real estate, and to create an additional constant to reflect the KI of the current purchase, as shown in the left panel of Figure 4.4. It has an additional advantage of a clearer distinction between data from both types of purchases. The effect of this choice on the GUI is shown in the screenshots below.

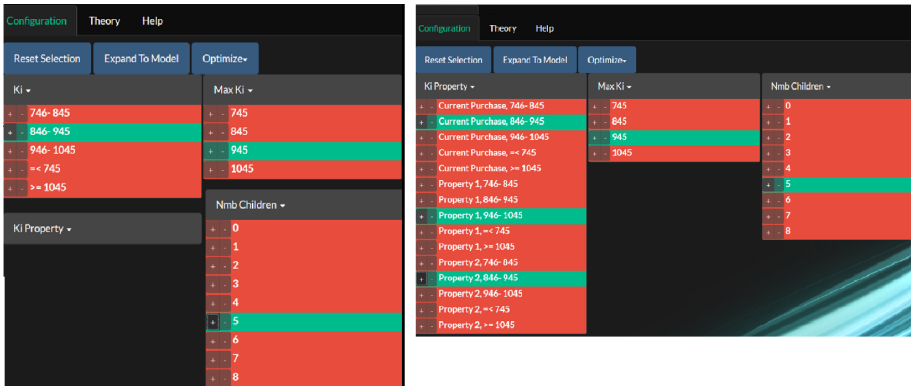


Figure 4.4: Translation of variables and impact on interface.

This results in a correct and concise representation of 11 articles of law, formalized in a knowledge base of 53 concepts, 6 constraints and 14 rules. Building this knowledge base required an effort of approximately 10 person-days. A significant part of this time was attributed to the creation of the set of symbols representing concepts in the domain (i.e. the *vocabulary*). To this end some analysis beyond the level of the DMN model was needed.

4.4.1 Evaluation

FO(·) proved to be expressive enough to formalize the entire decision model, and to create a term for the optimization of the taxes to be paid. We examine the maintainability claim of KBS with the effort necessary to update it to the changes in legislation enacted in 2018. These changes consisted of 5 new articles, making it the most significant change to real estate sales law since the transfer of jurisdiction from the national to the Flemish regional government in 2013. At the time of constructing the original knowledge base, the content of these changes was not yet known. Therefore, this provides a realistic test case to judge the maintainability of the knowledge base.

To give an idea of the required changes, we briefly discuss an example. In the original prototype, the *RegistrationType* is defined by a number of rules such as:

$$
\begin{array}{ll}
\{ & \mathrm{RegistrationType} = \mathrm{Social} \leftarrow \varphi_1 . \\
& \mathrm{RegistrationType} = \mathrm{Modest} \leftarrow \varphi_2 . \\
& \ldots \qquad \}
\end{array}
$$

After the amendments, new registration types needed to be added (e.g., *Monument*), while some disappeared (e.g., *Modest*) and others remained unchanged (e.g., *Social*). Thanks to the modular nature of rule-based definitions, these correspond to local changes that preserve the global structure of the definition:

$$
\begin{array}{ll}
\{ & \mathrm{RegistrationType} = \mathrm{Social} \leftarrow \varphi_1 . \\
& \mathrm{RegistrationType} = \mathrm{Monument} \leftarrow \varphi' . \\
& \ldots \qquad \}
\end{array}
$$

Updating the knowledge base required only 0.5 person-days, a fraction of the time required for the initial version. 16 of the original 53 concepts were removed and 18 new ones added; 11 existing constraints and rules needed to be updated or deleted, while 4 new constraints were added. Crucially, 9 of the 20 existing constraints/rules did not need to be touched at all. This demonstrates that the inherent modularity of the KBP indeed leads to significant advantages in practice.

4.5 Prototype

We use the FO(·) specification as KB in the interactive consultant to create a first prototype of the application.

- With *model expansion* all models that comply with the given information can be obtained. To handle the requirements of reasoning with incomplete information, we used the approach from [30].
- The inference task *propagation* is used to derive all consequences of the information that is already available. Based on this, the GUI shown in figure 4.4 is updated: mandatory values are highlighted in green and impossible values in red. As long as no colour is assigned, information relevant for this decision is missing.
- Another form of inference that was used is the *optimisation* of selected terms. In many cases buyers want to minimise current total registration duty payment. This can be done by minimising the term $(BaseAmount * TaxRate) - TotalPortability$. It is possible to create other terms to optimise other criteria.

This first prototype was used to solicit feedback on requirements of the application, as they were under-specified at the outset of the project. Feedback on the prototype led to the development of a new interface.

4.5.1 Advanced prototype

The new interface contains usability updates, such as information tooltips, custom input fields and a start screen with a limited set of predetermined *core* constants. We have also updated the visuals to make the symbols (✓ and ×) more prominent and less colour-based.[1]

A more important improvement is the clear distinction between *chosen atoms* (set by the user) and *propagated atoms* (implied by the chosen atoms). The interface visualizes chosen atoms by a ⟳-symbol to indicate that this choice can be reconsidered. Propagated atoms are visualized by question marks, indicating that they can be *explained*.

As before, the interface operates independently of the domain knowledge gathered in the knowledge base. This is still the case in the new interface. Additionally, a separate *comma separated value* file – denoted as *the CSV* – is required as input. The CSV contains some configuration options and meta-information on the symbols to be used by the GUI.[2] A first piece of

[1] The red-green colour scheme that we intuitively chose in analogy with traffic lights is not distinguishable for colour blind persons. Additional attention needs to be devoted to this when developing the prototype in an actual application.

[2] The CSV file was replaced with a json file afterwards, and is removed completely in the newest version of the interactive consultant. This information is now provided in the display section

meta-information is a priority value for symbols and atoms. This priority value determines which concepts are shown first to a user.

Other pieces of meta-information are information strings for symbols and atoms, which allow the GUI to elaborate on the intended meaning of symbols and atoms in the knowledge domain. In this application we used it to reference the applicable law article.

The most significant improvement is the application of the *relevance* and *explanation* inferences for interactive decision enactment.

Explanation To increase user confidence, it is important that the system is able to explain why it derived certain conclusions. Moreover, the user sometimes would like to flip a propagated atom's truth assignment. The explanation inference identifies the chosen atoms that imply a propagated atom, and allows to revise these choices. As input, it takes a theory T, a partial interpretation I and a propagated atom a. As output, it returns a least precise partial interpretation I^{expl} such that a would still be propagated by I^{expl}.

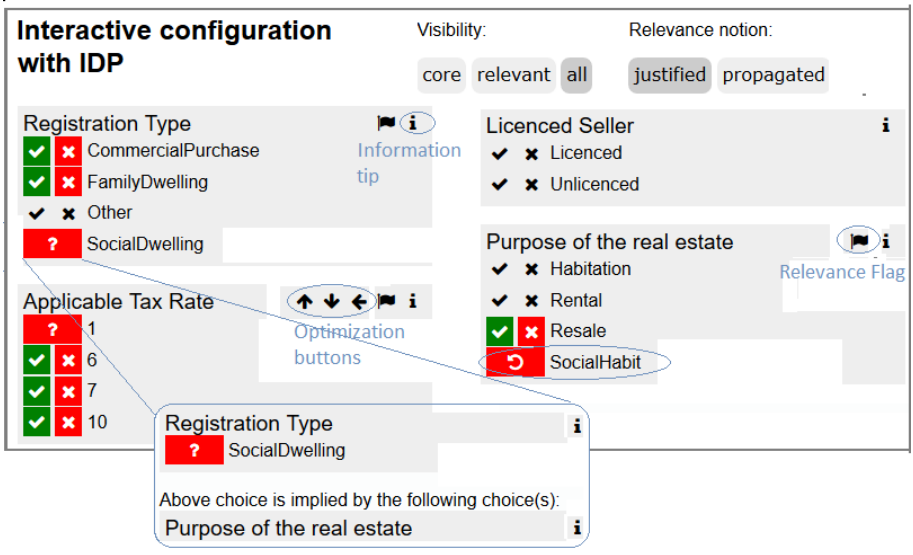


Figure 4.5: Relevance and explanation demonstrated in the new interface.

This is demonstrated in Figure 4.5. When a user clicks on the question mark of a propagated atom a, the system constructs the partial interpretation I^{chosen} from all the chosen atoms, and feeds a and I^{chosen} to IDP's explanation inference

together with the theory T containing all domain knowledge. The output then represents a minimal subset of all chosen atoms that still imply a's propagated value, which is presented to the user as an explanation for the propagation.

Relevance A key problem of the original prototype was that it encouraged notaries to ask irrelevant questions. For instance, the knowledge base included the concept of a *licensed seller*: only when the seller is licensed, can the property be eligible for a social registration. In particular, the definition of $RegistrationType$ contains the following rule:

$$\{RegistrationType = Social \leftarrow Seller = Licensed \wedge Purpose = SocialHabit.\}$$

Moreover, this is the only formula where the licensed seller concept is used. Once the notary has determined that the purpose of the real estate is not social habitation, there is no longer any need to determine whether the seller is licensed.

Our new prototype makes use of the relevance inference to avoid this problem. This inference takes as input a theory T, a partial interpretation I_p that is closed under propagation, and a set of *goal constants* C. Its output is a set of *relevant* atoms $a = v$ that can still affect the interpretation of the constants in C, given T and the information present in I_p. In our example, if I_p is such that $SocialHabit \in Purpose^{I_p}$ and $C = \{RegistrationType\}$, then the atom $Seller = Licensed$ is relevant, as choosing its truth value determines whether $RegistrationType = Social$. If $SocialHabit \notin Purpose^{I_p}$, then $RegistrationType = Social$ is false in all model expansions of I_p w.r.t. T, and $Seller = Licensed$ is therefore not relevant.

IDP's relevance inference is based on justification theory (e.g., [34]). The concepts we introduce in the following paragraphs are highly simplified versions of the original justification theory and of the implementation of the relevance inference that is available in our software tool.

The *dependency graph* of a theory T has all of the sub-formulas of the theory as its nodes and has an edge from each formula to all of its sub-formulas. In addition, for each rule of the form $A \leftarrow \varphi$, there is also an edge from the atom A to the formula φ. Intuitively, each directed edge from φ to ψ in this graph means that the truth of φ can be justified by the truth of φ. Finally, we also add each of the goal constants C to the graph and include an edge from each goal constant $c \in C$ to all atoms of the form $c = v$. The idea behind these edges is that value of the goal constant c is influenced by the truth of these atoms $c = v$. We denote the resulting graph by G_T^C.

Figure 4.6 demonstrates the dependency graph for our limited example in which the applicable rate is determined by the registration type.

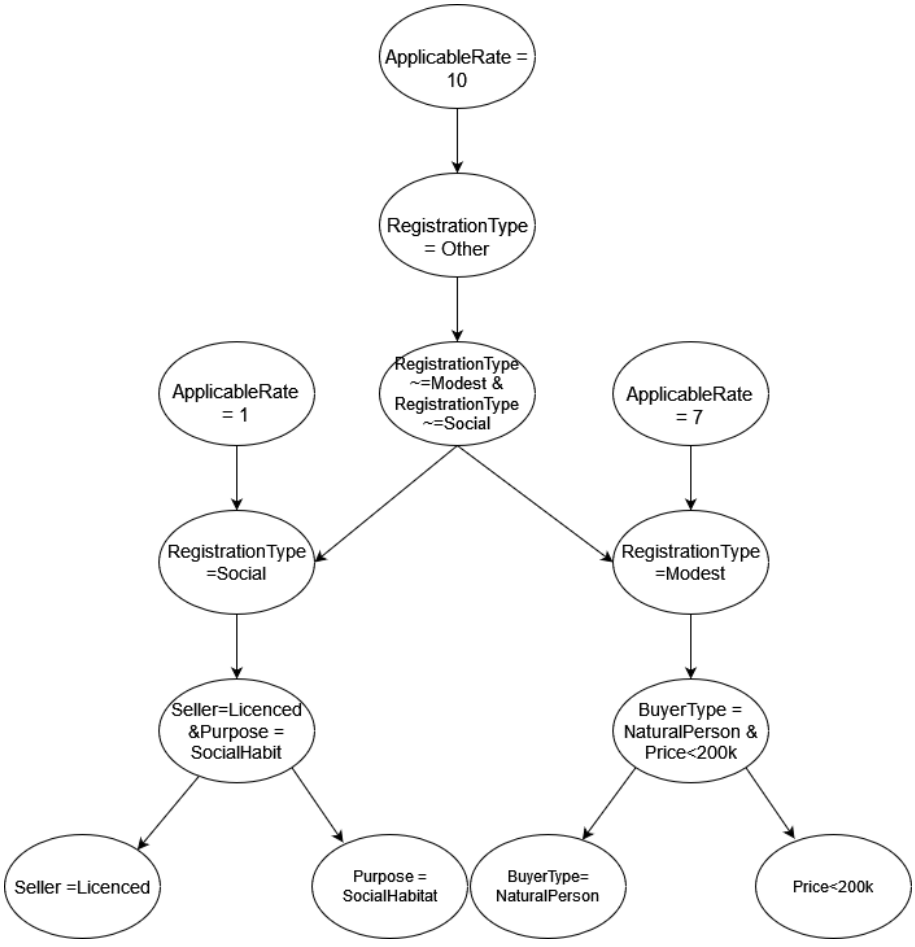


Figure 4.6: Illustration of a dependency graph.

We say that a formula φ is T-*determined* by I_p if either $I_p \models_T \varphi$ or $I_p \models_T \neg\varphi$. Finally, we define that an atom $c = v$ is *relevant* if it is not T-determined by I_p and there exists a path in G_T^C from this atom to one of the goal atoms, which does not traverse any node that is T-determined by I_p. In the interface, a relevant choice $c = v$ is highlighted with red and green buttons (that can be used to assign it true or false), while irrelevant choices can still be made, but the buttons are grey. In addition, the box for a constant c is flagged in

the upper right corner to indicate that at least one relevant atom over c still exists. Finally, it is also possible to hide irrelevant unknown atoms. Figure 4.5 shows the atom $Seller = Licensed$ is indeed irrelevant (for the goal constant $RegistrationType$) under the given truth assignment.

4.5.2 Evaluation of the prototype

Regardless of the implemented functionalities, one of the main concerns of the notary is that the application should be non-intrusive. Clear colouring in green and red blocks improves the ease of use, while the hidden background information ensures minimal cluttering of the interface. The explanation functionality was found to be useful to explain which choices led to the inquired result. The adding of information on which article of law the choices stem from, was suggested as a further improvement. Especially the relevance algorithm was much appreciated. This algorithm shows which choices still impact the final outcome after propagation of all known information. Whereas clients previously needed to fill in a five-page document with information that is sometimes hard to retrieve and/or superfluous, they now only have to provide essential information.

Further internal testing of the interface unveiled a difficulty with regards to what *ought to be* true and what *is* true [23]. We discussed above that the FO(·) specification contains a lot of definitions. These definitions have a certain hierarchy, in the sense that the highest level definition contains concepts, that are in themselves defined at a lower level. The notary select concepts at each level. If the value "true is assigned to a concept, and this concept is the result of a definition, propagation might assign a value to underlying concepts. These concepts ought to have this value in any model. However, nothing enforces this value in reality, and probably it is not even possible to choose the value of this concept in reality. An example may clarify this. A property may be bought by a real estate broker with the intention to re-sell it on short notice. With this in mind, the notary selects the buyer type "professional buyer". To qualify as professional buyer in this context, this person needs to register a statement of profession, and needs to re-sell at least three properties in the course of the next five years. Both conditions ought to be true. However, if the client was not aware of the obligation to file a statement of profession, this might be false in reality. To make this distinction in the application, the concept of advice and situation variables was developed by Carbonnelle et al. [23]. Advice variables are variables defined by the theory. If their value is propagated to true, they become true in reality. However, situation variables are determined in the environment. If they are propagated to true, the corresponding check is highlighted in orange. The notary should now verify the value of this concept in reality, and confirm it as true to continue. This distinction should be expanded

if legal applications for law-suits would be developed. Then distinction should be made between what ought to be true (from the theory), what is claimed to be true (evidence filed) and what is accepted to be true (evidence accepted). Future work should focus on different truth values and their implication for the development of a KBS.

4.6 Discussion of case study results

We created an elaborate DMN model that consists on one hand of DRDs to visualise the structure of the legislation, and on the other hand of decision tables that contain the decision logic as it is inherent in the articles of law. Following the guidelines of Bench-Capon the model was isomorph to some extent, although we added an additional variable (registration type) that allows users to refer to a specific tax discount. Although the model is executable, it proved mainly convenient as an intermediate model for the creation of a FO(·) KB, as the execution of the decisions in the DMN tool OpenRules does not behave as needed.

The fact that the analysis of the problem domain was already done with the DMN-tables allowed the modeller to focus exclusively on the formulation in the FO(.) language. Therefore the development of the IDP model was relatively easy. The GUI is derived automatically from the IDP model and requires no extra work. The interface in which the propagation of available information is visible, is perceived as very user friendly. The notary sees at a glance which information needs to be requested from the buyer, and which information is superfluous. Therefore the application provides useful support during client meetings. Further improvements in usability can be obtained by further exploring the possibilities of separated vocabularies. For the discounts for modest housing, *abattement* and portability, certain settlement restrictions are in place. These restrictions differ slightly for the different discounts. As a result three variables with almost the same name are created. By working with different vocabularies, groups of related variables can be grouped together, making it easier to visually relate the correct settlement variable with the intended concept.

Inherent to the concept of a KBS is the absence of a distinction between input and output variables. Besides the different inference possibilities this also means in this case that every variables can be determined by the user (as long as no other value is propagated). An example is the definition of a professional seller. If this is a large organisation or a frequent client of the notary, the notary probably knows it is a professional seller without checking the different conditions. However, for a new professional buyer/seller the determination of

the legal statute should happen by defining the prerequisite conditions. IDP allows notaries to choose if they want to select manually the value *Professional Buyer*, or if they want to check the preconditions.

The result of this case validates two central claims of the KBP. First, the effort to update the knowledge base (0.5 person-days) was very small in comparison to the effort to create the initial knowledge base (10 person-days), especially when taking the size of the legal changes into account. There are various factors that contribute to the limited time necessary to update the model such as increased familiarity with the domain and the existence of an ontology and a model, etc. It should be noted that it is thanks to the inherent modularity of the original model that the existing ontology and model can be re-used in the new model. Therefore it demonstrates the maintainability of an approach based on the KBP. Second, it also shows a first-time integration of the relevance and explanation inferences in an interactive application, and demonstrates their practical utility. This demonstrates the flexibility to use KBSs for different tasks. The resulting interactive decision enactment system is applicable to a wide range of applications.

Chapter 5

Legal applications

The previous chapter (Chapter 4) describes the case of the development of a prototype for the calculation of registration duties. In this chapter we seek to generalize the findings of this case. We do this by embedding it in the relevant literature, on the one hand, and analysing the available expert knowledge on the other hand. We start by giving a definition of a legal application and the kind of legislation we target. Then we discuss some characteristics of legal applications, the knowledge they contain and how it can be obtained from experts. Finally, we come to some recommendations and conclusions.

Contributions We contribute to the body of knowledge by analysing the types of knowledge present in or necessary to develop legal applications. We formulate recommendations based on the experience from our case study and triangulate it with literature.

Defining legal applications In this thesis we define a legal application as an application aimed at formalizing statutory law and supporting decision making based on it. Important in this is that we focus on statutory law. The legal domain is very broad, and encompasses different types of law, the most well-known distinction being the difference between common law as practiced in, e.g., the US and UK and civil law as practiced in, e.g., Germany, France and Belgium. Although some statutes also exist in common law systems, the role of case law is predominant, whereas codified statutes are prevalent in civil law [111]. The envisaged application is limited to support operational decisions, such as encountered frequently in administrative law or tax law. It is not suitable

for lawsuit advice, which involves other activities, such as finding precedent cases, presenting evidence (that might not be accepted), rejecting or accepting counter-evidence,...

5.1 Characteristics of legal applications

Rulemaking is a core task of government. It affects private and professional areas of citizens, companies and governmental actors [102]. Automating the rules offers a large potential for increased efficiency, service, correctness and consistency. Even a small efficiency gain has the potential of affecting a broad public because of the large application area of governmental rules. Moreover, governments do not escape the digitization trend [107]. With the need for correct decisions and individual's right to know how decisions affecting their life are taken [129], a KBS can offer many advantages.

Legal application requirements Important in legal applications is first the ability to give a correct decision. Moreover, to build user confidence it is important that users know how the application arrived at its conclusion. If they want to challenge it, for example because they have new information, it is important to know which information has (not) been taken into account. Finally, the legal domain uses a lot of different concepts that need to be defined in the application, and that risk to clutter the user interface. When one factor is set, a bunch of concept that depend on this factor, may not be relevant anymore, and should not be shown anymore to avoid this cluttering. Therefore an inference task that can determine the relevance of concepts, is typically useful for legal applications.

Knowledge sources The most obvious knowledge source for legal applications are of course law texts. However, law texts are often difficult to interpret by laymen. The government often provides a user-friendlier version on topic-specific websites. Although the information provided in such websites often lacks the level of detail needed for a KBS, it helps the knowledge engineer to gain insight in the domain. Also human legal experts are indispensable for the creation of a KB. They dispose of knowledge that is difficult to access for domain laymen, such as the structure of the applicable law, the priority of different regulations, rule content, rule interpretation and analoguous case knowledge.

Incomplete information Another distinctive element in legal applications, is that some of the decisions are taken at one point in time based on intentions, but

that they can be revised afterwards in case these intentions do not materialize. An example of this is the intention to establish one's primary residence in a dwelling. It is one of the conditions to pay discounted registration duties. The original decision to charge 6% registration duties, will be revised and a final duty of 10% will be levied in case the owner does not establish his primary residence at the property. In general, in legal applications decisions often need to be taken with incomplete information. This can take the form of intentions that may or may not materialize, but it can also be in the form of missing or outdated information.

5.2 Knowledge in legal applications

In line with the different knowledge kinds discerned by Liebowitz [90] we discuss the domain knowledge, inference knowledge and task knowledge needed to create a legal application.

Domain knowledge The largest part of domain knowledge in legal applications, can be found in legislation. Legal texts are structured in parts, books, titles, chapters, sections, subsections,... following guidelines that determine how law texts should be written and which parts and references are mandatory (see e.g., published principles of legislative technique [114]). Typically the base case is determined first in a separate article of law. It is followed by exceptions to the base case, then deviations from exceptions, and –if applicable- sanctions in case of disobedience. Each stipulation is put in a different article. The nesting in chapters, sections, etc. helps the reader to understand the text, and determines the content, e.g., in phrases such as "taxes imposed in this chapter" [108].

As it is written down in law books, the knowledge in legal application is for the largest part explicit. Nevertheless, some elements are implicit, e.g., when they are derived from analogous reasoning, or when principles are described instead of rules. Moreover, just because knowledge is explicit, this does not mean that it is easily accessible: it may be scattered over different sources of law, and the priority of sources is determined in yet other sources of law. An important part of expert's domain knowledge consists of prioritizing the different sources and principles to be used.

Specific for knowledge in legal applications is that a lot of jargon, definitions and concept overloading are used. Terms such as "incriminating title" (bezwarende titel in Dutch) are specific to the legal jargon, and hardly used in every day life. Definitions can be explicitely labeled as such in the law text, but they are also implicitely present throughout the text. Definitions make up a large part of

law text, as is demonstrated in the case described in Chapter 4. Unfortunately, even in the same chapter, the definition of a concept may vary from one article to another, causing concept overloading. An example is the definition of "first house" that was different in the calculation of abbattement than in the discount of registration duties. These characteristics make legal texts often difficult to read for laymen, and difficult to formalize in a KB. Moreover, typical for legal texts is their reference to other law books in constructions like "as stated in article ..."or "in accordance with article 32 or article 33 of the federal Code of Registration, Mortgage and Court Rights". Delineating the scope of the application and determining the appropriate level of detail needed for the KB is a challenge for legal applications.

Inference knowledge Inference knowledge refers to knowing how to execute a simple task, for example arriving to a conclusion by deduction. Statutory law that is targeted in our application, is naively thought to be limited to this deductive reasoning task: if the terminology in the statute is clarified, the user only needs to deduce the logical conclusions that follow from it. This view is an over-simplification, as in reality the interpretation of statutes is much more complex than this. It is better characterised by defeasible reasoning: based on some information a provisional conclusion can be drawn, but additional information may defeat this provisional conclusion and lead to another conclusion [87]. A classic example of default reasoning is the following: "If Tweety is a bird, one will assume that it can fly. However, if Tweety is a penguin, the first conclusion is disregarded and one concludes that Tweety cannot fly". Default reasoning is prominent in the legal domain, and especially in statutory or public law [87]. This is reflected in the structure of the law, where the general rule is very often the subject of one article, with the subsequent articles describing exceptions to it.

Task knowledge Task knowledge refers to the ability to execute a complex task by decomposing it in simple subtasks and selecting appropriate solution mechanisms. In legal applications, this means making a distinction between areas in which deductive reasoning is predominant, and areas in which default reasoning is more apparent. As a result, caution is advised when developing the "unhappy flow" of events, such as contradicting information that becomes available after a decision has been taken. The system should be able to deal with it in an apppropriate way.

5.3 Knowledge acquisition in the legal domain

Acquiring domain knowledge The two main sources of knowledge for legal applications are law books and legal experts. We used a combination of text analysis and interviews as knowledge acquisition techniques. The role of the legal expert is important to guide the modeller through different sources of law, clarify jargon and interpret vague statements. Legal experts are used to default reasoning and reasoning by analogy. The modeller on the other hand will try to decompose the domain in (often hierarchically) structured decision points. To reconcile both views and enable the expert to assist the modeller, using a visualisation of the structure of the law text can be very helpful. This visualisation can be used to structure the inteviews with the domain expert. A DRD is such a standardized visualisation. It can be created in discussion with the expert, or it can be created by analysing the division in chapters and sections of the law text, and validating this afterwards.

The development of a legal application is complex because of the complexity of legislation itself. A first formalization of the domain in a simple model allows to analyse the complexity of the domain, and use this simplified model as go-between experts of different backgrounds. In other words, this first simple model is an intermediate model. After the modeller understands the structure of the domain, they can use this intermediate model as basis to develop more complex models. This additional complexity can come from the use of a more complex modelling language. We found DMN to be a useful modelling notation for the intermediate model, as basis for the development of a FO($\cdot$) model. The tables that represent the decision logic, can be validated with the expert. This task will be far less complex for the expert, if the model adheres to the principle of isomorphism, meaning that its structure is closely aligned with the structure of the original text. Using DMN as intermediate model has the additional advantage that examples can be run to detect gaps in the model.

Acquiring inference and task knowledge For people with limited insight in the structure and functioning of the system, it is often difficult to express how it should behave. A prototype can be used to express ideas (e.g., one can just point to the screen and explain what one expects to see), and to probe for information that is not easily remembered. The knowledge engineer should make sure to continue asking at a detailed level what the expert exactly means.

5.4 Recommendations to develop legal applications

The creation of the KB for a legal application is challenging from multiple points of view. The delineation of the scope is one of the most important challenges, as it should not only be limited in width (which law articles to include), but also in depth (how detailed should the used vocabulary be). This challenge is linked to the complex formal language and jargon. Moreover, legal reasoning for statutory law is characterized as default reasoning, which goes against the intuition that legal reasoning should be deductive.

Based on the experience in the registration duty case study and the specific nature of legal knowledge we formulate following recommendations:

- Use the help of a legal professional. Although a lot of legal knowledge is explicit and accessible to any citizen, there also exists a range of less accessible or intangible knowledge to which legal professionals have acces.
- A structured interview with a legal expert is a good starting point. During this interview the width of the scope should be determined (identify articles of law), as well as the structure of the law text. Keeping in mind the default reasoning, the interviewer can probe for this with questions such as “What is the default/most common case”, “What are the exceptions to this case”. As there are probably exceptions to exceptions, try to limit this in the first run to one level of exceptions.
- Interatively create a DRD based on the explanation of the expert and the structure of the law book. Use it to further delineate the scope, confirm your analysis of the structure and focus discussions in a next round.
- Create an intermediate model in a suitable, simple enough language. The choice for this language should be based on its simplicity and ability to be understood by experts from different disciplines. DMN, as a user-friendly notation, is very suitable for this. As each article of law typically contains a different stipulation, as a main principle each one of them is formalized in a separate decision table. Tables can still be merged afterwards, e.g., on the level of the base case or above. By proceeding this way, the modeller will create an isomorphic model (i.e., following the structure of the law book), which facilitates user validation of the model.
- Although the use of first hit tables is generally discouraged, they are a good way to reflect certain parts of legal knowledge.
- Also within articles, knowledge may be hierarchically structured. E.g., besides the core of the provision, they may also contain definitions of concepts. New tables should be created for each new level of information.

- Because of the large number of concepts and the overloading of concepts, it is important to create a glossary of the vocabulary to minimize the risk of mistakes.
- Use a prototype to elicit the functional requirements of the system. Ask detailed explanation on the purpose of required features and try to think of alternative interpretations thay may impact the development of inference algorithms. Keep in mind that there might be a difference between what should be (according to the model) and what is (in reality).

5.5 Conclusion

In this chapter we generalised the findings of the case study described in Chapter 4 to legal rule applications. Although statutory law is often seen as certain, clearcut and subject to deductive reasoning, this is an over-simplification. Legal experts use defeasible reasoning to draw conclusions in light of the available information, keeping in mind that over the course of time new information that necessitates a revision of the original decision, might become available. This also impacts knowledge acquisition. Where developers are accustomed to using decomposition, pattern recognition, abstraction and algorithms (i.e., computational thinking), legal experts focus on exceptions, rare cases and details. Empathy with and understanding of this legal reasoning will bring the knowledge engineer a long way when developing a KB for legal applications. First, the knowledge engineer can use it to steer interaction during expert interviews. Instead of gradually refining the model, it makes more sense to dive into one exception, and work them through in what seems to be an unstructured way. It will be up to the the modeller to find the inherent structure in this information, such that it can be modelled adequately. The second result from understanding this, is the impact on the choice for modelling tools. From a validation point of view, it is advantageous to create an isomorphic model, i.e., the structure of the model should ressemble the structure of the base document. DMN combines two advantages in this respect. The DRD is able to catch the highly-structured nature of legal texts, and reflect it in a readable manner. The first hit decision tables reflect the default reasoning. Hence, it unites the predominant way of thinking of both knowledge engineer (decomposition in DRD) and legal expert (default reasoning in first hit tables). Even though DMN tools might not offer the required functionalities, the described advantages make it very useful to create an intermediate model. Third, it will allow to better interpret the user's concerns and functional requirements, and probe further for their intentions. The use of a prototype will also be helpful in this regard.

Chapter 6

Case: Seal design

In this chapter we describe the second case study in which we investigate the fitness of KBSs for the creation of product configuration systems (PCSs). The case is part of a multi-year project that aims to automate product configuration for standard products and provide support for the design of non-standard products. The project was split in six application areas, of which we modelled one. Colleague Bram Aerts modelled the other five application areas, and we could rely on his expertise during the course of the project. This case has been described in a journal paper together with Bram Aerts and Joost Vennekens, titled "Knowledge-based decision support for machine component design: a case study." [1]. The cDMN modelling was presented at RuleML2019 [1].

Contributions Throughout this case study we developed the knowledge articulation method and the cDMN notation. This chapter demonstrates the actual use of the method. This novel approach differs from the existing approaches by its focus on design decisions, the central role of the product engineer and the use of a single model. We developed both DMN (Case 1) and cDMN (Case 2) models, we identify strengthts and weaknesses of both methods and demonstrate the difference between both notations on a part of the case. We contributed to the development of a generic design support tool.

6.1 Introduction

This case describes the design, implementation and evaluation of two PCSs for the configuration of highly specialized products. The products consist of a number of different components, that come in various variants and sizes, and can be produced with various materials. The product can be used in six application areas that necessitate distinct designs. Customers order a product for a specific set of requirements, such as operating temperature range and pressure, size, etc. Typically, the customer is an engineer from another production company. Understanding and explicating the customers' needs is therefore not an issue in this application, in contrast to typical configuration problems [146]. The focus of the applications is the core design process that the company's product engineers follow to decide on the general design of the product, its specific components and their materials. At the start of our collaboration with the company the expertise of the engineers was required for the majority of client's orders. During the project we created two kinds of PCS: six make-to-order (MTO) PCSs – for the standard designs of each application area respectively –, and one engineer-to-order (ETO) PCS that abstracts and unites the engineering knowledge of the different areas in one configurator for non-standard designs. The difference between MTO and ETO is that the former works with existing designs (of which each component needs to be made specifically for the order), whereas the creation of the design itself is part of the latter.

Standard Designs Standard designs are designs that are used in standard operating conditions: moderate temperature, pressure, size, etc. If the customer's requirements could be fulfilled by one of the company's standard solutions that was part of their Visual Basic (VB) tool with a Microsoft Access database, the sales staff that received the incoming request, autonomously handled the request. Sadly, this tool only covered a small subset of the standard designs, and extending it was a difficult and resource demanding task. Moreover, because of the unstructured way in which the knowledge in this tool was represented, it was difficult to maintain.

Non-standard Designs Requests that fell outside the VB tool, 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 well-known circumstances, characterized by a standard temperature range, standard size range, standard pressure range, etc., this was a routine job for the engineer. If the product fell outside these well-known areas, the design was more challenging and required 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 actual design process.

6.2 Case 1: DMN-based make-to-order PCS

6.2.1 Requirements

The requirements of the company with regards to the standard products were well defined. First, they wanted to replace the VB tool that they were using with a solution that would be well-maintainable, preferably by the engineers themselves. This is a key requirement, as the creation of PCSs is very resource intensive and the company wants to make sure that this investment will be put to good use over the next years. Second, the scope of standard products that was part of the VB tool needed to be enlarged to encompass all standard designs. Third, the knowledge acquisition exercise needed to reveal expert's knowledge with the aim to store and safeguard it to prevent possible knowledge-losses associated with staff movement. Fourth, during the exercise engineers from different international branches were asked to align their way of working to impose standardisation. Fifth, the PCS needs to be integrated with other company systems to automatically create drawings and a bill of material.

6.2.2 Knowledge articulation workshops

As the maintainability of the application is a key concern for the company, we developed the knowledge articulation method that is in the first place useful for the knowledge acquisition phase of the project, but that also enables the domain experts to maintain the KB themselves. For the MTO PCS we choose DMN as the unique modelling tool because of the ability of DMN to be understood by business users (in this case, the product engineers) and the executability of the decision model.

To create a DMN model, we organised multiple diagramming workshops with the involved parties. Each workshop spans two to three days and results in an initial representation of the design process for a clearly delineated application area. Participants include three or four design engineers from different locations worldwide, the innovation manager, the project manager and one external knowledge engineer to guide the workshop. This approach offers a number of advantages.

- The involvement of multiple participants stimulates the exploration of different approaches instead of blindly adopting the approach of one engineer or location.
- The face-to-face time allows intensive discussion necessary to agree on an aligned approach.
- Each workshop focuses solely on one specific application area, which helps to keep the discussion focused.
- From a change management perspective, the involvement of participants reduces their resistance to the new application and increases visibility of the project.
- As layman in product design, the knowledge engineer asks "trivial" questions that help to clarify the engineers' design processes and to ensure that they are all on the same page.

Each workshop starts with an introduction to DMN, after which the knowledge engineer guides the domain experts through the modelling process. First, the standard operating conditions are identified to delineate the scope of the application. After this, we typically start by constructing a DRD 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 in the DRD. The role of the knowledge engineer is guiding the conversation, visualising the diagram as explained by the engineers, probing for further information in case of gaps and refocus discussion. A workshop results in a formal representation of the engineers' knowledge. As the DMN model is executable, this provides us with an initial prototype of a decision support system for that particular application area. This prototype is then presented to the design engineers for evaluation. Depending on how close to reality the preliminary model is, this evaluation can be done by e-mail or in another workshop. Based on the feedback, the model is refined. This process is repeated until all parties agree that the model is correct.

Evaluation of DMN for a make-to-order PCS

Following the methodology outlined above, we have completed the knowledge extraction process for six different application areas, producing a total of approximately 190 decision tables. In each of the application areas, one or two tables were pure data tables, consisting of numerical data for dimensioning the components. Due to similarities between the different application areas, limited reuse of decision tables was possible. The size of the extracted tables ranges from 1 to 38 rows, with an average size of 5 rows and 3 input conditions. To

validate our DMN models, 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. We used OpenRules[1] to implement the model. It generates the correct design in all of the test cases. Computing a design takes about 0.3 seconds single core on an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz. This is sufficiently fast for business use.

The knowledge articulation approach in which the engineers create the diagrams themselves, proved successful. As is common in the graphic ideation phase, the workshops made participants think about how they come to a design in a given situation. Consequently, it helped them to better understand and standardize their design process. The engineers appreciated the intuitive nature of the DMN standard for the modelling, as it seemed to fit well with the engineers' 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. 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. Eventually, the design engineers were comfortable enough with the notation to leave certain decision tables as "homework" after 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.

However, occasionally the engineers do not follow a strict bottom-up decision procedure. For instance, to ensure that the component stays in place, either a thicker spring or a closed design type can be used. A thicker spring with an open design type is preferred, but this is not always feasible. In particular, in cold circumstances, the product may shrink to such an extent that the open design fails, even with a thicker spring. However, to know whether this is the case, the shrinkage of the product has to be computed. Because this depends on the used materials and the precise layout of the components, this computation can only be done at the very end of the design process. Therefore, the engineers first assume that the open design type will suffice. They completely design the product based on this assumption and then compute the shrinkage. If it turns out that the shrinkage is too big, they backtrack and restart the design process with a closed design. Such a "guess and check" procedure cannot be elegantly represented in DMN. This lack of expressivity has been recognized earlier, e.g., by Car [21], who points out that the lack of expressivity does not

[1] http://openrules.com

hamper automated decision making as such, but does force the modellers to be creative with the logic representation. This typically reduces the value of the model as a source of documentation and as a means for internal standardization, and this therefore undesirable. In our case, the limitations of DMN forced us to adopt an "err on the side of safety" approach to handle the issues concerning shrinkage. While our solution is 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 greatly reduce the legibility of the DMN model.

The lack of expressivity of the standard becomes clear when interwoven, optional or cyclical decisions need to be modelled [99], or when background information or constraints need to be modelled. This typically forces one to mix different kinds of knowledge within a single table, which reduces the maintainability of the table. Suppose that a supplier changes the price of a material. This may have an impact on the preferred components of the design, as the engineers try to design the product at the lowest cost and some components require more material. However, if we do not know that the outcome of the Materials table is composed of the constraint that a material needs to be able to withstand the operating conditions and the preference for the cheapest material, it is impossible to determine the impact of this change. A representation that separates preferences from constraints would not have this problem. The inability to represent constraints and background information leads users to develop additions to the standard, e.g.; decision logic for background information in [19], or a constraint addition in OpenRules [48]. We have developed cDMN to overcome this difficulty [3]. Besides these downsides of the DMN standard itself, the limited possibilities of the available inference engines also limit the use of the notation. They currently support only a single inference task, namely that of computing the "output" decision variables given all the input variables.

6.3 Case 2: FO(.) based engineer-to-order PCS

The first case study focussed on the creation of six PCSs for MTO products that are used by the sales staff. This case study is done in the same company, in the same department and for the creation of the same classes of products. However, in this case study we focus on the creation of ETO products: non-standard products that need to be designed specifically to meet the client's needs. As we already have a lot of information on the product and the decision procedure that is used for standard products, we use this as input for the creation of the ETO configurator. The decision procedure is governed by a number of physical constraints (e.g., a material $M1$ can only be used in temperatures $\leq 100°C$)

and preferences (e.g., material $M2$ is preferred over material $M1$, e.g., 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' existing design process. These preferences and constraints contain more information than is explicitly present in the design procedure, because they also explain *why* certain designs are not chosen. 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. We will discuss this in Subsections 6.3.2 and 6.3.3. First we look at the requirements of the ETO configurator.

6.3.1 Requirements

The most important requirement of this PCS is that it should assist product engineers with the design of a wide range of non-standard products. As the product is non-standard and concerns a new design, the system should not return one product as solution. Rather, it should be able to depict a solution space, from which the impossible designs were excluded. The tool should contain a lot of information, i.e., not only the fact that a certain part is selected, but also the reason why this is the case. Engineers should be able to optimise the design following the client's needs. As the consultation of historical designs is an essential part of expert's design process, the tool should promote the extraction of relevant historical designs. As for the other PCSs, the tool should be easy to use and well-maintainable by domain experts themselves. We used a two-step approach: first we elicitated physical constraints of the design, afterwards we focussed on preferences.

6.3.2 Eliciting physical constraints

Although some information about physical constraints on components and materials is documented, the majority of this information exists only in the engineers' head. Without a structured approach, the elicitation of this information is difficult. We use the decision logic models (i.e., the tables) to structure the process. Different topics are addressed in a meaningful order and we can be sure that no relevant constraints are 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. We organised 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 multiple 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. 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 6.3 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 provides 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 possibly help us revise our initial guesses.

From a theoretical point of view, it is interesting to note that, although most of the physical constraints have a similar if-then structure, two kinds can actually be discerned. Some of the constraints are descriptive in 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.

$$DesignType = Open \Leftarrow ReleasePressure. \tag{6.1}$$

$$Location = PressureAccumulating \Rightarrow ReleasePressure. \tag{6.2}$$

Both kinds of constraints are used next to each other. The practical relevance of this distinction for our current application seems limited, but it might become more important as technology evolves. Although we have not considered it during development of the application, some of the functionalities might only make sense with a specific kind of constraint. For example, the deactivate constraint functionality, in which some of the constraints can be ignored until a later stage in the design. In a pressure accumulating environment, there should always be a way to release this pressure. This constraint cannot be ignored, and even with advances in technology, this will probably remain a ground rule. The fact that open designs release pressure, might change if the overall composition of the open design changes. Note that the difference between "what is" and "what should be" is similar to the registration duty case in Chapter 4.

6.3.3 Eliciting preferences

Typically, there are 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 optimisation 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 that 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 optimisation 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 standardisation and reduce the impact of individual preferences of engineers. At the same time, however, the engineers should retain enough flexibility to take the wishes of individual customers into account. This dual aim necessitates a dual elicitation approach. First, for the company's desire to standardize products, a new workshop with the innovation manager and project manager was organised. The result of this workshop is a three-levelled optimization framework (see Figure 6.1).

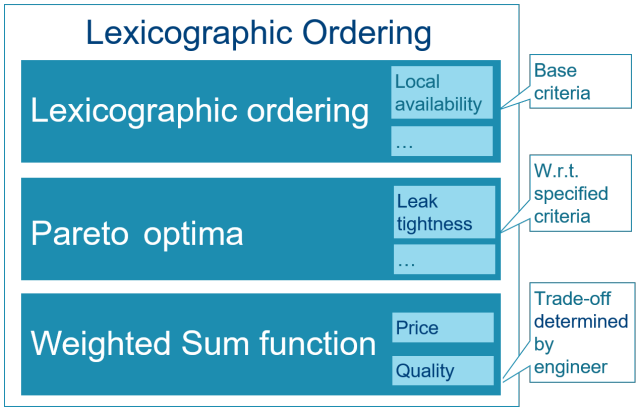


Figure 6.1: Optimisation framework.

The highest level consists of a lexicographic ordering of the solution space, according to a sequence of criteria $m_1, ..., m_n$. This means that all solutions that have a non-optimal value for m_1 are immediately discarded if there exists at least one solution with the optimal value for m_1. Among the remaining solutions, those with a non-optimal value for m_2 are also discarded, and so on. The first criteria $m_1, ..., m_{n-2}$ are all atomic criteria, such as the local availability of materials. 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 $m'_1, ..., m'_k$. 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 without making at least one of the other criteria worse. 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 their preferences. With a slider the engineer may indicate the importance of a limited number of numerical 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. In summary, our preferences are of the form:

$$lexico\big(m_1, ..., m_n, pareto(m'_1, ..., m'_k), max(\sum_i \alpha_i m''_i)\big) \tag{6.3}$$

where

- The m_i and m'_i are partial orders.
- The m''_i are numerical criteria.
- The α_i are weights to fine-tune the importance of each of the numerical criteria.

This framework contains the company's standardisation preferences: the first two levels are generic and even transparent for the engineers. The third level gives the engineer flexibility, but only on the criteria that are selected by the company.

The second form of preference elicitation, is elicitation of client's preferences. This is done by the engineers in their day-to-day work, through frequent contacts with the client, by soliciting feedback on proposals etc.. This is the kind of domain knowledge that we earlier reffered to as *customer preference knowledge*. Engineers can implement these preferences by interactively finetuning the weights of the weighted cost function.

6.3.4 cDMN

The use of constraints and preferences has a clear advantage over the DMN decision 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 this knowledge to be used in multiple use cases.

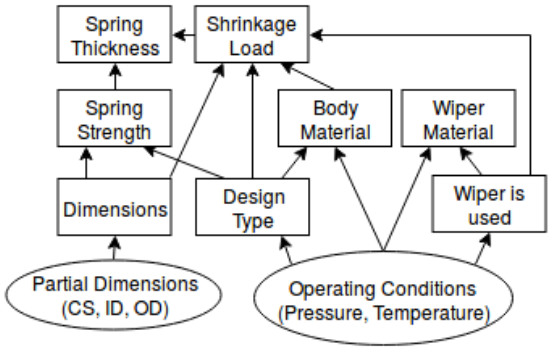


Figure 6.2: DRD model of component design decisions.

Although the modular sentences in FO(·) are well-readable when trained in it, for domain experts it is typically not possible to understand the knowledge captured in the model. This opens the way for misunderstandings and prevents the domain experts from working on the model without the presence of a knowledge engineer. This goes against the idea of the knowledge articulation method, in which domain engineers should be able to control the domain model themselves. Therefore we use the cDMN notation to model the domain with preferences and constraints. To demonstrate the difference between a DMN model and a cDMN model we discuss a highly simplified model of our product design model in both notations.

DMN model

Figure 6.2 shows our DRD, which captures the general structure of the decision logic. It starts from customer requirements at the bottom and has the decisions that must be made at the top, with certain subdecisions in between. Each rectangle corresponds to a decision table, which can be found in Figure 6.3. As all of our tables use the *U(nique)* hit policy, the conditions on the input columns in different rows must be mutually exclusive.

To design a component, the engineers typically follow different steps: *1. There are 3 relevant* dimensions*: the outer diameter (OD) of the component, its inner diameter (ID) and its cross-section (CS). The customer provides 2 of these and the engineer computes the third* ($CS = (OD - ID)/2$)*.* The corresponding DMN table 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.* This decision fits well within the mold of a DMN table.

3. Based on the temperature restrictions and design type, the "best" material *is selected for the body.* When representing this decision in a DMN table, we lose the distinction between physical constraints (i.e., the fact that some materials simply cannot cope with certain temperatures) and preferences (i.e., the fact that some material is not chosen because a cheaper material is available). Because this distinction is lost, it is not clear how to update this table when, e.g., the price of a certain material drops: there may be cases in which this material was originally not chosen because it was too expensive, but we cannot discern them from cases where the material was not chosen because it would melt.

4. 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.

5. Based on the selected design type, the spring strength *of a standard spring is computed. If this is enough to cope with the expected shrinkage, the standard spring is selected. Otherwise, the engineers switch to a thick spring and recompute the spring strength of the design.* Since DMN does not allow to "recompute" a value, the "Spring Strength" table computes both the spring strength that the design would have with a standard spring and that with a thick spring. The "Spring Thickness" table 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 counterintuitive, because in reality it is the thickness of the spring that determines the design's spring strength, not the other way around.

6. If, for an open design, even the thick spring does not provide enough spring strength, the engineers' final option is to switch to a closed design type, but since this is an initial choice upon which all further choices are based, this means that the engineers essentially start over the entire process from scratch. If the thick spring's strength does not suffice, our DMN table will fail to assign a value to "Spring Thickness" (i.e., it will be `null`). This will alert the user that something is wrong, but the backtracking step of redoing the entire design process with a closed design is not possible in DMN. For this reason, we have implemented a workaround in which the "Design Type" table is overly cautious: in low temperatures, it will always choose a closed design, even in cases when an open design could suffice. This is sub-optimal, but avoids the possibility of failing the spring thickness check at the end.

Dimensions

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

Body Material

U	Tempe-rature	Design Type	Body Material
...	...	...	...
5	[-50, 80]	Closed	M1
7	(80, 150]	-	M3
...	...	...	...

Spring Thickness

U	Shrinkage Load	Spring Thickness
1	<= Standard Spring Strength	Standard
2	(Standard Spring Strength,Thick Spring Strength]	Thick

Design Type

U	Pressure	Tempe-rature	Design Type
1	<= 100	-	Closed
2	(100,150]	<= -50	Closed
3	(100,150]	> -50	Open
4	> 150	-	Open

Spring Strength

U	Design Type	Standard Spring Strength	Thick Spring Strength
1	Closed	15	20
2	Open	7	11

Shrinkage Load

U	Design Type	Shrinkage Load
1	...	f(Cross Section, Material of Body, Temperature)

Figure 6.3: Decision tables for component design

Shrinkage Check

E*	Shrinkage Load
1	< Spring Strength

Design Type

E*	Pressure	Design Type
1	> 150	Open

Shrinkage Load

U	Design Type	Shrinkage Load
1	...	f(CS, Material of Body, Temperature)

Spring Strenght

U	Design Type	Spring Thickness	Spring Strength
1	Closed	Normal	15
2	Closed	Thick	20
3	Open	Normal	7
4	Open	Thick	11

Dimensions

U	CS
1	(OD - ID)/2

Operating Conditions

E*	Material of body	Temp
1	M1	[-100, 150]
2	M2	[-120, 80]
...	...	...

Material of Body

E*	Design Type	Material of Body
1	Closed	Not(M2)

Figure 6.4: Combination of constraint and decision tables

cDMN model of product design

The engineers preferred way of working can be represented more naturally in cDMN. The model can be found in Figure 6.4. We discuss some features of our constraint table representation of the running example in more detail.

Dimensions DMN tools use decision tables in a strictly feed-forward way, deriving outputs from inputs. By contrast, when we feed the constraint tables

into a FO model generation tool, this can use the constraint in different ways. For instance, it can use the *Dimensions* table to derive eithe CS, OD or ID from the other two.

Design Type The only physical design type constraint is that a closed design is not possible for pressures larger than 150. Nevertheless, the corresponding table in our original DMN representation (Figure 6.3) also contained the temperature as an input column. This was to avoid running into cases in which the shrinkage cannot be addressed by simply adding a thick spring to an open design. By contrast, our constraint representation states directly that a design type must be chosen in accordance with the constraints expressed in the *Design Type* table, the *Shrinkage Load* table and the *Shrinkage Check* table, avoiding the need for the original work-around.

Materials Each material has a minimum/maximum temperature, defined in the *Operating Conditions* table. It states that, if a material is used, the operating temperature should be within the allowed temperature range of that material.

In comparison with FO(·), the cDMN model is expressive enough to reflect the constraints of this case. Moreover, 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).

DMN versus cDMN

When we compare the DMN model with the cDMN model, the first observation is that the cDMN model is larger than the standard DMN model. The simplified DMN model consists of only six decision tables, whereas the cDMN model for the same scope consists of eight tables that determine the physical constraints. To express possible preferences, additional tables should be created. However, the larger size of the cDMN model is compensated for in terms of practical use:

- It is not necessary to determine a decision procedure prior to creating a cDMN model.
- The cDMN model contains substantially 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, but contains no information about alternative solutions, nor the reason why a value appears in the output column.

- Consequently, the knowledge in the cDMN model can be used in more 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. The outputs in a DMN table combines preferences and constraints in one output. If a preference changes, the output needs to be analysed to determine if and which adjustments are necessary.

Evaluation of the constraint approach

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 optimisation, allowing a solution that is optimal in a specific situation to be chosen from this set of solutions. Moreover, it allows to distinguish between physical constraints and subjective preferences. This makes the model easy to adapt to changes in the data (such as availability of materials), without requiring a review of the entire decision process. The constraints can also easily be reused in new application areas for which there is no complete model available.

However, we also found a disadvantage of the approach. The constraint approach is less comprehensible for domain experts. We have developed cDMN because we believe it is quite understandable for untrained domain experts, but it is still more complex than the simple table-based DMN format. This is probably due to the inherent complexity of reasoning with constraints compared to rules-based reasoning. In a DMN decision model, there is always a clear link between input and output, and outputs can only be defined in one table. This makes it easy for domain experts to understand the link between variables. In constraint reasoning, a single output may be affected by numerous constraints in different constraint tables. 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.

6.3.5 Prototype

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. With an evolutionary prototype, feedback can be solicited from the users and integrated in the next development phase.

We used the Interactive Consultant as prototype and developed it into an interactive tool to support the design process of product engineers.

Existing functionalities

In the next paragraphs we describe how the existing inferences were applied in our specific case.

- Propagation: 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.
- Explanation: This functionality is particularly important for the user acceptance of the proposed product design.
- Model expansion: In the specific context of our use case, this inference was not found to be very useful.
- Optimization: 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: This functionality was found useful because it prevents the engineers from having to make choices that are no longer relevant to complete the design.

Additional functionalities

Even though the Interactive Consultant was not specifically designed for our product design use case, it proved very useful. After testing the first version of the tool with the engineers, we found it was missing some functionalities for our case.

Comparison The engineers wanted to be able to *compare* multiple designs. Because of the large number of possible designs and materials, it is difficult to know the properties of each design by heart, especially for less experienced

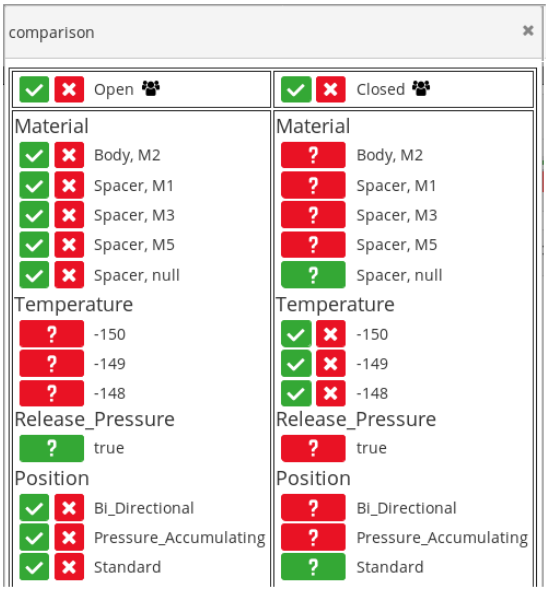


Figure 6.5: Comparison between an open and a closed design in the interactive consultant.

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 is in doubt between two options, it offers a quick way to see the implications of both.

Figure 6.5 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. In the example, the truth value of $ReleasePressure$ is ct in the three-valued structure of an open design, while it is cf in the three-valued structure of a closed design. In other words, pressure cannot be released in a closed design, while an open design always releases pressure.

The $ct/cf/u$ values are displayed in the same way as on the main screen, i.e., ct and cf values are shown respectively as green and red boxes. As the screen shows, 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.

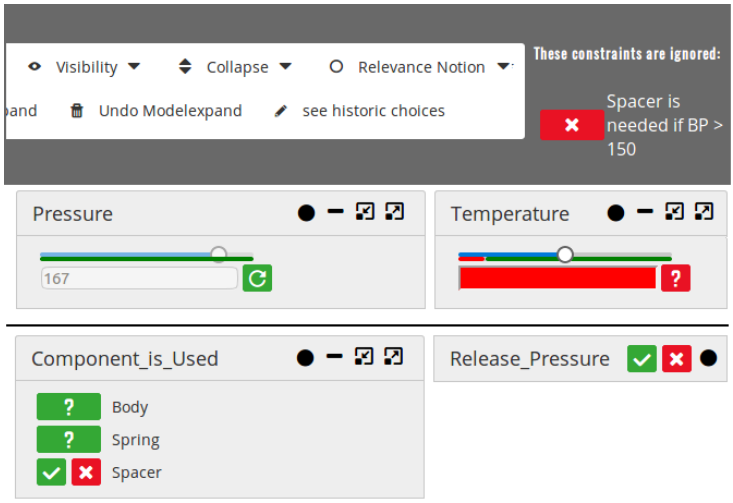


Figure 6.6: Visual representation of when a constraint is deactivated in the Interactive Consultant (detail of upper right corner).

Extended explanation The Interactive Consultant already offered the ability to explain why a certain atom was propagated. Such an explanation consisted of the set of choices made by the user that imply the atom in question. However, for the engineers, it was not always clear why these particular choices had an effect on that particular atom. Therefore, we extended this functionality to also identify the relevant constraints. This additional information allows the engineers to better comprehend the reasoning of the system.

To build an explanation, we use essentially the same method as was used originally. First, 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 determines the smallest subset of constraints that make a problem unsatisfiable.

As most engineers lack the ability to read constraints as they are formulated in the KB, we allow the KB maintainer to provide a readable annotation for each constraint.

Deactivate constraints It is possible that an experienced engineer has more thorough knowledge of the domain at hand than the knowledge base. This might become apparent when the system draws an incorrect conclusion. If this occurs, we provide the possibility to temporarily *deactivate a constraint*.

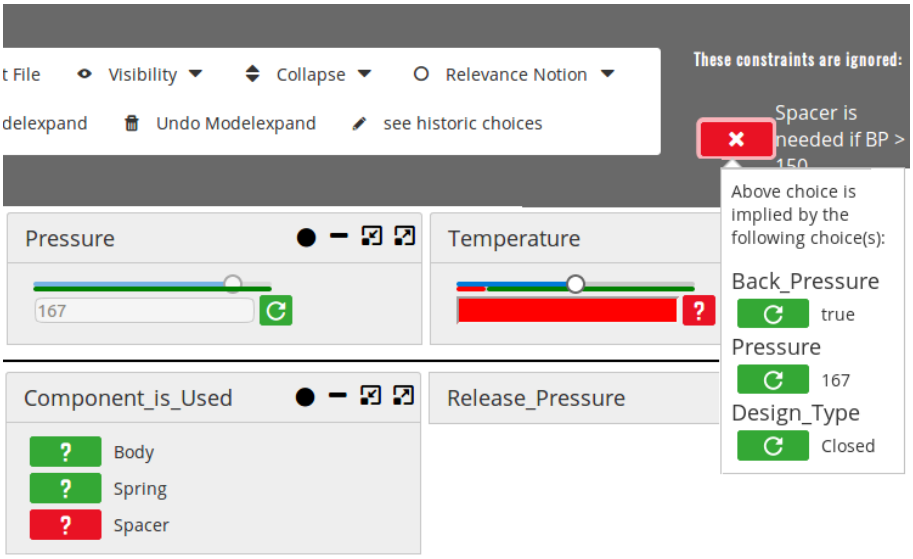


Figure 6.7: Indication of a constraint that cannot be reactivated in the Interactive Consultant (detail of upper right corner)

Because of the modular nature of the constraints in the KB, the IDP system can still draw conclusions from the remaining knowledge. This functionality enables engineers to experiment with out-of-the-box designs.

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. Once a constraint has been deactivated, it is important to visually indicate this, as can be seen in the upper right corner of Figure 6.6, so the engineer does not forget about it and possibly creates an erroneous design.

Reactivate constraints After having deactivated a constraint, the user might eventually want to reactivate it, e.g., because they have finished their experimental try-out design. Reactivating a deactivated constraint 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 6.7.

Provide Feedback When an experienced engineer is using the tool, they 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 flag that constraint or atom for review. The system then logs (1) the element that is flagged, (2) the current state of the user interface, which allows the problem to be reproduced, and (3) a textual description of the issue provided by the user. The KB maintainer then reviews the reports and updates the model if needed. This feedback mechanism ensures that the KB will be kept up-to-date, provided that experienced engineers consistently provide feedback when they encounter such imperfections.

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

Using Historical Data

Throughout the years, the company has 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 were made before. In addition to the constraint-based presentation of the product design knowledge, this historical information can also be very useful to support the engineers in their design tasks.

In a separate project, an algorithm was developed that is able to analyse the design features in the technical drawings and to measure similarity between designs [132]. We use this similarity measure to find designs that resemble the current state of the configuration process, and present the most similar ones to the user. The properties of each of these designs is listed, as can be seen in Figure 6.8. Additionally, a summary of all applicable solutions can be found in the right-most column.

Prototype evaluation

Procedure During the research project, engineers from the company were solicited multiple times to evaluate the tool and provide suggestions for improvement. So, over the course of several months, they got familiar with

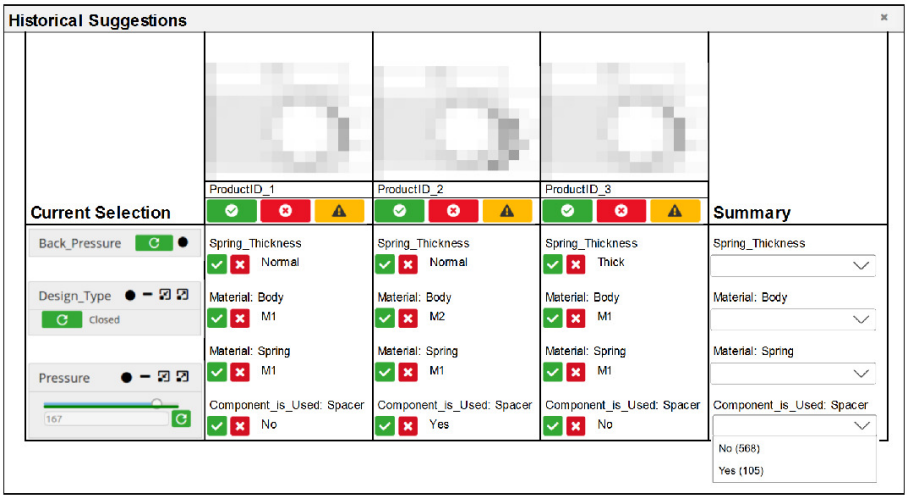


Figure 6.8: Browsing historical products in the finalised prototype (truncated image).

bits and pieces of the interface and its functionalities. After incorporating their suggestions in the Interactive Consultant, we organised an evaluation workshop to find out if the project goals were met. For the company the final goal of the project is the successful implementation of the application and its actual use in practice. As the implementation in production is still on-going, the latter is difficult to measure in practice. Therefore we looked for an alternative evaluation. Turan et.al. propose the model as pictured in Figure 6.9. The unified theory of acceptance and use of technology is an important part of this overall model. It was originally proposed by Venkatesh et. al. [141], who quotes four factors that determine the intention and usage of applications: performance expectancy, effort expectancy, social influence, and facilitating conditions (Figure 6.10). Our questionnaire consists of three parts: the expected usage of the tool, its expected benefits and its potential users. The first two parts aim to get a clear picture of the performance and effort expectance of the application, while the third part relates to the social influences and facilitating conditions. It is to be noted that our questionnaire does not cover all the aspects of the unified theory of acceptance and use of technology, because this would also cover mediating factors such as gender, age, and voluntariness of use, which are factors that do not fall within our influence. Our questionnaire focusses on how characteristics of the system influences the different factors. Five engineers with different levels of exposure to the tool and experience in the tested application area, were invited to use the tool.

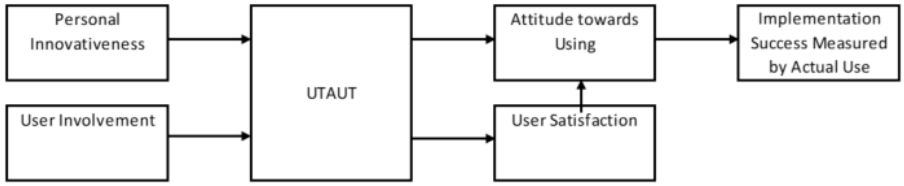


Figure 6.9: Expanded unified theory of acceptance and use of technology by Turan.

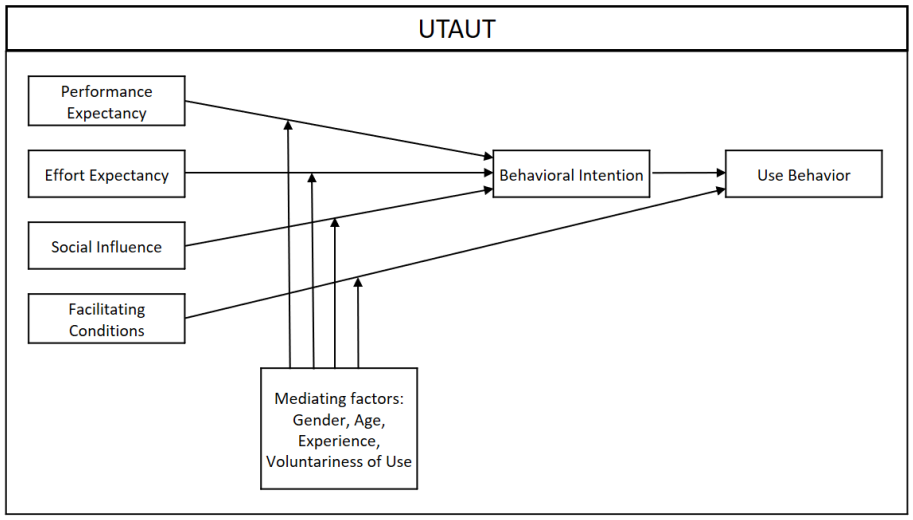


Figure 6.10: Unified theory of acceptance and use of technology.

Their design experience ranged from a couple of months to multiple years. Some of the 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. A researcher then 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. Once they had familiarised themselves with the tool, they received an assignment, i.e., a set of requirements for which they had to design a product 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 results are described below.

Usage of the tool The tool was developed to support engineers in the design of non-standard products. In our evaluation, we wanted to check whether the engineers found it useful for this purpose. In their answers, the engineers indicated they 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 specialised 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 this case, the tool might be useful to challenge their own assumptions and ensure standardisation. On the other hand, for designs with complex geometry, even after using the tool, the engineer would still need to make manual calculations to fine-tune the dimensions of the components. In the follow-up discussion, the engineers indicated that they saw the manual calculations as inevitable, and not as a missing functionality 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 for double checking personal expertise.

Analysis of product failures was also mentioned as a potential application for the tool. It could be useful to enter the details of the failed product and the operating conditions at the time of failure into the tool, and check which constraints are violated. However, a specific functionality for this would be more convenient.

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 incentivises its use and stresses its importance in promoting consistent designs and validating one's assumptions.

Benefits of the tool The second part of the questionnaire focussed 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 were more responsive.

For some tasks the tool is useful and convenient, but the gain is diffucult to quantify. 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 machine learning 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 also 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 continuous learning about designs and their constraints, and promotion of a standard to judge design quality.

Target audience of the tool The usefulness of the tool might be different for engineers with different levels of experience. The interviewed engineers think the tool will be as useful for more experienced engineers as it was for them, and even more useful for less experienced engineers. In the discussion that followed, it was said 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 also 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 Overall, the engineers had a positive reaction 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 engineers appreciated in particular the combination of a knowledge based approach that lines out the constraints of the design, and the possibility to review historical designs. Additionally, the tool is not only relevant for the design of a specific product, but can more broadly 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 tool meets the requirements that were identified at the beginning of the project.

6.4 Conclusion

In this chapter we described two case studies that we executed at the same company. Both cases concern the construction of a PCS. While the first one is a MTO configurator, the second one is an ETO configurator. Both PCSs have different requirements and necessitate a different approach.

The MTO PCS is a deterministic rule-based system. For this we successfully used a knowledge articulation approach with DMN as unique language. The ETO is a constraint-based system. To expand the existing KB, we used again workshops, but this time with cDMN as modelling language. The cDMN model can automatically be translated to FO(·). This FO(·) specification is used as the KB in the Interactive Consultant as first prototype. This prototype was used to solicit functional requirements of the system, that were subsequently incorporated in new prototypes. After a couple of runs, this resulted in a powerful and positively evaluated design tool.

In both PCSs the knowledge of the product engineers plays a central role. By using the knowledge articulation method, this central role is confirmed and supported. The use of user-friendly modelling languages allows experts to keep the PCS up-to-date.

Chapter 7

Product configuration systems

In the previous Chapter 6 the cases of two PCSs at an international manufacturing company were described. In this chapter we generalise our findings by situating them in existing PCS literature. We start by defining and characterising PCSs. An existing PCS development framework is discussed. Next, we describe the different kinds of knowledge in a PCS. In this, we focus on the complexity of the knowledge, which is claimed to be very characteristic for PCSs. We amend existing complexity measures to be able to compare the complexity of our PCS with other applications. Then we focus on the knowledge acquisition. The last sections gather recommendations to develop PCSs and conclude the chapter.

Defining product configuration systems The cases described in Chapter 6 are examples of PCSs. Although the term PCS is commonly used, there does not exist a single, fixed definition. What is relevant is that the aim of the system is to support users to compose an end product of different components. We define a product configuration systems as a system to support the design of products that consist of multiple components in multiple variants by formulating constraints on how these parts can be combined to arrive at a product with the required characteristics. A typical example is a car configuration in which users can decide on which variant of certain parts they want to include in the design (e.g., leather chairs), or if they want to add parts to the configuration (e.g., a tow bar). The choices users make, results in a custom-made product, for which the price depends on the different customisations.

This kind of business-to-consumer product configurators and the trend towards mass customisation mutually support each other. Mass customisation is the xenogamy between mass production and customisation, that tries to accommodate individual customer's needs while at the same time addressing a large market [152]. The level of customisation is indicated by the customer order de-coupling point CODP, that is the point that separates the internal, planning-driven parts of an organization, from the customer-oriented parts that allow for differentiation [10, 91]. In the stream from raw materials to the delivery of the product, the CODP represents the point from which the product is linked to a specific customer [91]. In increasing order of customisation, following CODP strategies can be discerned in a manufacturing company:

- In a Make-to-stock (MTS) strategy, standardised products are produced, and companies focus on stock management. Given sufficient stock, products are immediately available.
- Assemble-to-order (ATO) companies create a stock of parts that can be combined to the clients' needs. If all parts are available, the composition of this product is relatively fast. Car configurators are situated in this strategy.
- Make-to-order (MTO) products are existing products that are produced upon the client's request. As the product is only produced after ordering, the client might need to wait a considerable time. The first case of Chapter 6 falls in this category: after selecting the correct design and properties of the product, the drawing is automatically created and passed to production.
- Engineer-to-order (ETO) products are products that are designed according to the client's needs. Before producing the product, the characteristics of the design need to be laid out. The application that we created in the second case of Chapter 6 is an example of an ETO configurator: it assists engineers to create a new design for a non-standard product.

The different strategies are visualised in Figure 7.1. The strategy determines which kind of PCS is needed, and who will be using it. Today, the largest part of PCS is found in expert and business-to-business markets [116]. Mass customisation and product configuration are not limited to material products. Also immaterial products, or services, that exist of multiple component in multiple variants that can be combined under restrictions to obtain a valid end product, could be configured by using a PCS. For example, think about how the possibilities to compose a study program have increased the last decades. An

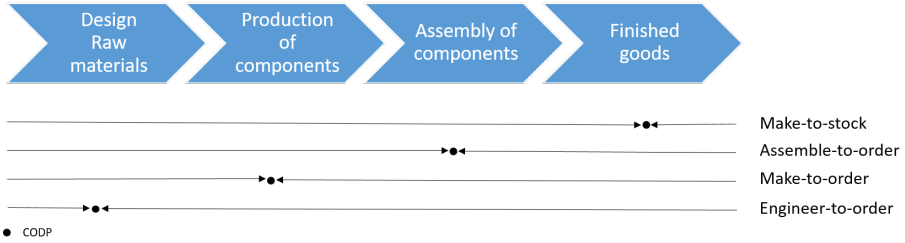


Figure 7.1: Visualisation of CODP in a production chain (adapted from [10]).

example of an immaterial product configurator for this is the course selection demo on the IDP website[1], that demonstrates how a study program can be composed of a number of mandatory and optional courses. The intangible nature of services could make the configuration of a complete and correct specification more difficult in comparison to tangible product configuration [65].

Contributions The distinction between different kinds of PCSs has been recognized in the literature before. In this work, it is the first time that the impact of this distinction on the development of the system is investigated and described. We provide a first time evaluation of the newly developed knowledge articulation method.

To situate our contributions to the field of PCS we use the review of Zhang et al. [151]. They analyse several aspects of product configuration, including knowledge acquisition, knowledge representation, configuration explanation, the design of new components, and reconfiguration. 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. An exception to this is the work of Shafiee et al. [121] on the framework for the knowledge acquisition in configuration projects. Our work contributes in this area by proposing and evaluating the knowledge articulation method. A second aspect which according to [151] 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. In other words, most focus is on make-to-order (MTO) and assemble-to-order (ATO) configurators, and less on engineer-to-order (ETO) configurators. Our constraint-based approach is able to provide

[1] https://krr.gitlab.io/demos/

useful information to the engineers in cases that fall outside the scope of existing solutions. Third, [151] also identifies several ways in which additional forms of inference might be useful to provide functionality other than suggesting a design. For example, they identify 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 [33], in which different logical inference methods can be applied to the same knowledge base in order to implement different functionalities. Indeed, in Section 6.3.5, 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 has already been considered in the context of this system [143, 148]. The IDP system therefore provides a suitable framework to implement these different tasks.

7.1 Characteristics of PCSs

The appeal of a PCS is its huge potential for benefits if implemented properly. Well-documented benefits are more correct creation of a wide range of product specifications, and massive time reductions in man-hours, lead time and quotation time, all of which have a possitive impact on the sales [64].

It is generally accepted that PCSs stand out from other kinds of systems because of the complexity of the knowledge they contain [121, 64]. This has some practical implications that characterise the development project of PCSs and make it different from other software projects. Shafiee et al. describe this extensively in their work on configuration projects [122]. To start with, scoping the project is difficult [122]. The scoping process should not only take into consideration the existing product variants that need to be included, but because of the given new possibilities that evolve quickly, the decision to include or exclude variants needs to be repeated over and over again. Moreover, the complex knowledge can be modelled in different ways, so the decision on which models will be created in the course of the project, is part of the scoping decision. The PCS can be enriched with price and production information, and integrated with other applications such as drawing tools, etc. Also from this point of view the scoping of a PCS is challenging. Second, the models underpinning the KB contain very detailed infomation and, –if not co-created by them–, need to be validated by domain experts. This means that there needs to be an intensive communication between the knowledge engineer and the domain experts, that goes beyond typical communication between experts and IT developers [122]. Third, the maintenance of PCS is crucial to enjoy the anticipated benefits, as

outdated PCSs will simply not be used anymore [120]. Although all outdated systems will lose their utility, maintenance is particular important for PCSs. The parts and material to create a variant of a product tend to evolve rapidly. Moreover, because outdated knowledge of any part of the system impacts the entire design procedure, small knowledge gaps might make the entire application superfluous. Therefore a PCS risks to become quickly outdated. Specific for PCS is that the rules and constraints that govern the design of the final product are established by the company. This is an essential element to discern PCS from third party applications as described in Chapter 10: although both kind of applications aim to accommodate customer's preferences, in PCSs the KB is constructed by the company, whereas it is constructed by the client in third party applications. This distinction will be discussed in detail in Section 10.2.

When looking at the difficulties associated with the introduction of PCSs, the top three difficulties are resistance to use the configurator, missing support from top management and knowledge acquisition difficulties, followed by complexity due to a lack of product range overview ([84], p.208).

7.1.1 Development of PCSs

Knowledge acquisition is stated as one of the most difficult activities in PCS projects [84]. Multiple knowledge management frameworks to ensure knowledge control in projects exist [121]. However, these frameworks do not take into account the very knowledge intensive nature of PCSs. To fill this gap, Shafiee et al. [121] developed a knowledge management framework specifically dedicated to use in PCS projects. It consists of four phases, each with its own tools. In the first phase the scope of the configurator is determined. The second phase is labelled the knowledge acquisition phase. It proposes a narrow interpretation of the activity as the categorization of knowledge (explicit versus tacit), and the listing of knowledge sources (e.g., internal documentation, company experts). The third phase consists of product modelling and knowledge validation. The proposed modelling techniques encompass product variant master (PVM), collaboration cards (CRC) and a unified modelling language (UML) notation. Different kind of models may exist for the same product. A phenomenon model is intended to be used by domain experts, the information model is developed for use by IT staff, and the computer model can be used by the computer ([44] in [71]). The fourth step jumps to documenting and maintaining the knowledge.

The framework is characterised by the following observations: i) it consists of multiple, well-delineated and connected activities; ii) various models and techniques are used to different ends; iii) the core of the knowledge base consists of product models; iv) the knowledge engineer plays a pivotal role in

the knowledge management process; and v) it focuses on domain knowledge management, and less on inference or task knowledge.

7.2 Knowledge in PCSs

Configuration knowledge of a company is present in the heads of –often geographically and organisationaly dispersed– domain experts or R&D departments. In the introduction we described the three kinds of knowledge as identified by a.o. Liebowitz [90] and Hall [61]. In this section we describe each of these categories in detail.

Domain knowledge In the context of PCSs domain knowledge encompasses knowledge on the existing parts, how and to which end they can be combined together. Moreover, knowledge on the specific characteristics and content of each part is needed. Domain experts typically dispose of some of this information, while some knowledge is available in technical product descriptions of the components, and yet other knowledge needs to be discovered by tests in the R&D department. Another form of knowledge that is typically less associated with engineering skills, but is nevertheless important to create a product according to customer satisfaction, is customer preference knowledge. A given set of requirements could be fulfilled by multiple variants of the product. The customer knowledge of the engineer will lead him to propose a specific product, for example the one with the best quality, the cheapest, the most user friendly, the fastest to deliver,... This is intangible knowledge, that is typically gathered throughout multiple interactions with the client.

Inference knowledge In the context of a PCS, inference knowledge means knowledge on the simpler subtasks that need to be executed in the course of a configuration activity. In the cases described, some of these subtasks are: ensuring the design can withstand the pressure, ensuring the design can withstand the temperature, ensure the product fits in the cavity, etc.

Task knowledge Apart from this domain knowledge, engineers also posses the skill to combine different forms of reasoning to solve the complex task of creating a product. In the case, this means all of the subtasks have been solved and resulted in a final product. Although experts use this knowledge every day, it is typically difficult to explain to someone else because it is the kind of subconscious knowledge that was automated by experience [67]. It is up

the the knowledge engineer to make this knowledge explicit. The final form of knowledge needed for the creation of a PCS is knowing what the PCS should look like and what it should be able to do. This depends on the user of the configurator: is it the end-client who will use it directly? Or will sales staff translate client's needs to propose a product? Or will the product engineers themselves use it?

7.3 Knowledge acquisition for PCSs

In spite of the reported advantages of the proposed knowledge management framework by Shafiee et al. [121], we see challenges that are not addressed by it. Different models are constructed for the same product. Each new model gives rise to modelling costs, and creates a source for potential misinterpretation. Moreover, although the attention for documentation in theory allows for maintainable software, this is not necessarily the case if there is no maintenance process in place. The framework of Shafiee et al. suggests that new knowledge acquisition, modelling, validation and documentation steps should be followed [121]. This would require the help of a knowledge engineer for each change, which makes each change a burden. Finally, although the elicitation of functional requirements is recognized as an important part of PCS projects, this is nowhere reflected in the framework.

As an alternative we propose the knowledge articulation method that we described in Chapter 3. The specific role of domain experts for PCS development has been stressed by leading authors in the field [121, 64, 122, 71, 84]. Moreover, also the documentation of the underlying models is very important in PCSs [120]. This aligns well with the proposition of the knowledge articulation method that experts should have a central role in creating, using and maintaining the KB. The use of a single model enhances communication between domain experts and IT staff, and provides a documentation of the system at the same time. As the underlying product knowledge of configurators is often dispersed over different experts in a company, knowledge articulation workshops with multiple experts and the use of a diagramming method that may overcome language differences (not only differences in the actual natural language, but also differences in jargon that might exist over different company sites), are seen to deliver large added value.

In the studied case, the involvement of multiple experts also proved to have an impact from a change management point of view. First, because the company has different sites worldwide and experts from all over the world were involved in the workshops, this resulted in considerable travel expenses. The company

also incurred an opportunity cost for the time the experts could not spend on their other tasks. The fact that the company is willing to bear these costs, signals the importance of the project. Second, because more people are involved, it shows that their opinion is important, and supports the claim that the project is aimed at making the engineers' lives better. The involved participants have the opportunity to learn about the tool. They are likely to become change agents.

7.3.1 Evaluation of knowledge articulation method

Our proposed knowledge articulation method has multiple advantages. In many decision automation projects the responsibility for the decision is handed off to the programmer or an intelligent algorithm. In cases where the correct interpretation and application of rules is essential, a company may suffer considerable damage if a system proposes a wrong decision. With our method, the expert maintains ownership of the KB, and hence of the decision. The occurrence of mistakes is less likely, as there is no translation from one interpretation to another. The maintenance of the KB, an important issue for KBSs which operate in ever-evolving environments, is a lightweight procedure that users can perform continuously. Moreover, by using a language that is easily understood by experts, it is easier to involve more experts with different levels and areas of expertise in the project, as they have a unique common language. In traditional methods, domain experts have a rather passive role: when asked, they share their knowledge, after which they wait for the KBS to be ready. The knowledge engineer on the other hand, carries the heaviest workload, with intense communication with IT and domain experts, understanding and modeling products with which they were previously unfamiliar, change management, etc. Both actors get a different role in our method. Domain experts are more involved, are able to concisely express themselves and are continuously involved in an active role as owners of the KB. The focus of the knowledge engineer shifts to planning and organising, ensuring training and facilitating the process.

A possible downside of our method is that experts need to master the modeling language. This can be countered by the careful selection of the language. If the language is simple and intuitive, the learning effort is minimal. Moreover, if a standard language is chosen, experts might already be familiar with it. Thus, the additional effort compared to the advantages over the entire project might be negligible. Another possible disadvantage is the novelty of our proposed way of working. Finding a balance between the expectations towards experts and knowledge engineers may be challenging at first. It may also induce resistance from the impacted parties. However, both experts and knowledge engineers

have an important role to play in the project, and in our case studies we notice that the rapid result often convinces skeptics.

The use of the interactive consultant proved effective as evolutionary prototype to elicit task and inference knowledge. The workshop format offered a lot of advantages because almost all needed knowledge was present, different views could be aligned, and seemed to have supported change management.

7.4 Recommendations

MTO PCS contain deterministic information. This relates well with a rule based approach.

Before starting a PCS development project, the purpose of the application needs to be clarified. Following questions should be answered:

- What strategy will the PCS support? ATO and MTO strategies contain deterministic, rule-based knowledge; ETO strategies require constraint based knowledge. This difference will result in different knowledge acquisition and system development approaches (see below).
- Is the project about automating the as-is situation, or does it also aim to rationalise/align/standardise company practice? In the first case, one or a few experts can be interviewed one-by-one, in the latter case it is advantageous to involve a couple of experts from different areas in workshops.
- Will the system be used for the training of novices? In that case the explanation of results is an important feature of the system. An interactive application with the possibility to generate explanations, such as the interactive consultant, is preferrable. For training purpose the system should at a minimum be able to explain its decisions. To understand how a complex decision is impacted by its subdecisions, a training system should allow the processing of partial information. This allows the trainee to investigate stepwise the decision process. The current DMN based OpenRules tool has a limited support for explaining decisions, and cannot show partial results. The IDP system allows the flexible use of the KB, also for training purpose. Also the feature to de- and re-activate constraints allows a novice to play with the system and gain insight in the domain.
- Is the documentation of knowledge important? At what level? Both DMN and FO($\cdot$) models centralise domain knowledge. However, the constraint

based knowledge in FO(·) and cDMN allows a more fine-grained level of knowledge by making a distinction between physical constraints and preferences.

Assemble-to-order and make-to-order PCSs The rule-based knowledge can be obtained by using the knowledge articulation method with DMN as unique modelling language. In our case, the MTO PCS was used only to automate design selection. The DMN model was implemented in OpenRules, and integrated in the existing production chain to allow straight-through processing of standard designs. It is for training purposes possible to automatically translate the DMN model in a FO(·) specification, to use the interactive consultant as training tool. The remark to keep in mind in this context is that a rule-based model is less informative than a constraint-based approach.

Engineer-to-order PCSs Starting from a deterministic model, the underlying constraints and preferences can be obtained through interviews or workshops. The use of an evolutionary prototype is helpful in acquiring task and inference knowledge. As the interactive consultant allows to build an evolutionary prototype, it is a time and cost efficient way to develop an application.

In both cases the involvement of multiple experts facilitates change management, because participants act as change agents and the involvement of multiple key personnel in the organisation signals the importance of the project for the management.

7.5 Conclusion

PCSs are a specific kind of KBSs because of the complexity of the involved knowledge, the fact that this knowledge is fragmented in the company over multiple domain experts, and the fast development of varieties. This has important consequences with regards to the creation, usage and maintenance of PCSs. As none of the existing knowledge acquisition methods was able to support the entire knowledge acquisition process, including the maintenance of knowledge, we developed a method and the supporting tools that can act as a single PCS development method. To stress the central role of the domain expert in this method, we labelled it the knowledge articulation method, meaning that we give domain experts the tools to express their knowledge in such way that it can be understood and used by the entire company.

In this chapter we evaluated the use of the knowledge articulation method and made recommendations for the development of PCSs, subject to the kind and purpose of the application.

Chapter 8

Case: Doctor scheduling

In this chapter we discuss the case of an emergency service doctor schedule. The case was executed with the help of dr. Crits and the regional hospital Heilig Hart in Tienen. Colleagues Jo Devriendt, Simon Vandevelde and Pierre Carbonnelle discussed and developed features of the Interactive Consultant to make an interactive scheduling tool of it. Kylian Van Dessel created an API between Excel and IDP.

In the next section we introduce the case and the requirements of the planner. Afterwards we discuss the use of DMN and cDMN. Section 8.4 describes the FO(·) model of the domain. In this section we start by describing the high-level characteristics of the case, after which we focus on specific problems, such as the use of a calendar and the creation of a cost function to prioritize preferences. The schedule that we created could never completely fulfil the requirements of the planner, due to difficulties to elicit all requirements for the planning and the change of their importance. To overcome these difficulties, the application should allow for a high level of interactivity, that allows the planner to adjust costs or force the planning. The role of interactivity with regards to the planning is discussed in Section 8.5.

Contributions We demonstrate the suitability of the IDP system to create a scheduling application by creating a model of the scheduling requirements and developing an interactive scheduling application. We describe the different steps that have been taken in the development process, and tested resolution strategies for different kinds of problems. We know from previous research that there is a large gap between academically available scheduling solutions and their use in practice. In this case study we describe the difficulties to

develop a practical scheduling application and how we come to the solution of an interactive scheduling system.

8.1 Introduction

This case concerns the scheduling of doctors in existing time slots. There are three shifts per day, seven days per week, that reflect the legally obliged presence of doctors in emergency services. Exactly one doctor needs to be assigned to each shift. Not every doctor has the same work regime: some doctors work one day per week, others work half time, still others work full time. Weekend shifts are assigned proportionally. Doctors are employed as independent contractors, meaning that there few labour laws, such as maximum daily working hours, rest times, overtime,... are applicable. To allow for a tolerable work-life balance the planner wants to take into account individual preferences and habits with regards to the regime. However, when needed doctors are flexible to fill up gaps.

Only four constraints were identified by the planner as compulsory or hard constraints:

- If a doctor works the night shift, they cannot work the early or the late shift next day.
- If a doctor works on Saturday, they must also work on Sunday.
- Each shift needs to be assigned to exactly one doctor.
- Each doctor can only be assigned to one shift per day.

Besides the hard constraints that determine possible schedules, there are plenty of soft constraints that determine the quality of the schedule. The fulfilment of these preferences is not mandatory, but desirable to obtain a schedule that meets the perceive quality standards.

- Proportionate assignment of number of shifts, weekends, night shifts depending on the work regime of each doctor.
- Maximum number of (night) shifts for specific doctor
- If worked during the weekend, have the next day off.
- Preferred working days and days off.
- Group together the days off of any doctor per week.

Some of these preferences are very unique for a single doctor (e.g., a doctor that works a maximum of two night shifts per month), other preferences are common for all doctors (e.g., proportional assignments of shifts).

Finally, as it is clear beforehand that the schedule will not be able to respect all of the constraints and preferences, the impact of the violated constraints should be balanced over different doctors.

8.2 Requirements

The purpose is to create a schedule as if the human planner had made it, but without the time the planner would need to be spend on it. The planner currently uses Excel to visualize, communicate and follow-up on the schedule. Trust needs to be build before switching to another system will be considered.

8.2.1 Schedule requirements elicitation

The first step to start the creation of a scheduling tool, is to understand the required characteristics of it, and the planner's current way of working. We met several times with the planner. During the first meeting, the basic functioning of the IDP system was shortly explained. Most of the meeting was used by the planner to explain how she currently proceeds. After this first meeting, some prototypes of a planning were mailed to the planning, and the resulting feedback was processed in the next version of the KB. Even after several runs, the outcome was not as desired by the planner, and new preferences kept surfacing. Therefore we needed to alter the purpose of creating the perfect schedule to providing a system that would interactively support the planner to create a balanced schedule.

8.3 Modelling the domain

After the first meeting with the planner, the acquired knowledge was modelled. We tested both a tabular and FO($\cdot$) modelling.

8.3.1 DMN and cDMN

The creation of a DMN model to represent the scheduling constraints proved impossible. DMN decomposes a decision to be taken in smaller sub-decisions until the only input of a decision are external inputs. This leads to a strictly hierarchical and non-cyclic decision graph, as represented in DRDs. For the scheduling problem, this kind of decomposition does not make any sense, as the decision for a shift assignment depends on shift assignments on other day.

It is possible to model the constraints of a schedule in cDMN. This results in a model with 31 tables, including tables with definitions, such as the In Weekend table, count tables, such as the Number of Shift table, constraint tables such as the Number of Shift per Doctor table (Figure 8.1).

In Weekend			
U	Shift	Weekday of Shift	Shift is In Weekend
1	-	Saturday	Yes
2	-	Sunday	Yes
3	-	not(Saturday Sunday)	No

(a) In Weekend decision table

Number of Shift					
C#	Doctor	Day	Shift	Planning of Shift	Number of Shift per Doctor per Day
1	-	Weekday of Shift	-	Doctor	1

(b) Number of shift decision table

Number of Shift per Day				
E*	Doctor	Day	Shift	Number of Shift per Doctor per Day
1	-	-	-	$\leqslant 1$

(c) Number of shift constraint table

Figure 8.1: Extract of tables from the doctor scheduling model.

With the use of cDMN it is less straightforward to create an overview of the decision logic as it is to create a DRD of a DMN model. A cDMN model is typically larger than a DMN model, as one specific rule may be the combination of multiple underlying constraints. Moreover, in constraint reasoning multiple factors may impact each other, which means that circular patterns will appear in the diagram. Another complicating factor is the absence of a distinction between input and output variables. The development of a simple yet expressive visualisation for a constraint based problem is future work.

8.4 FO(·) model

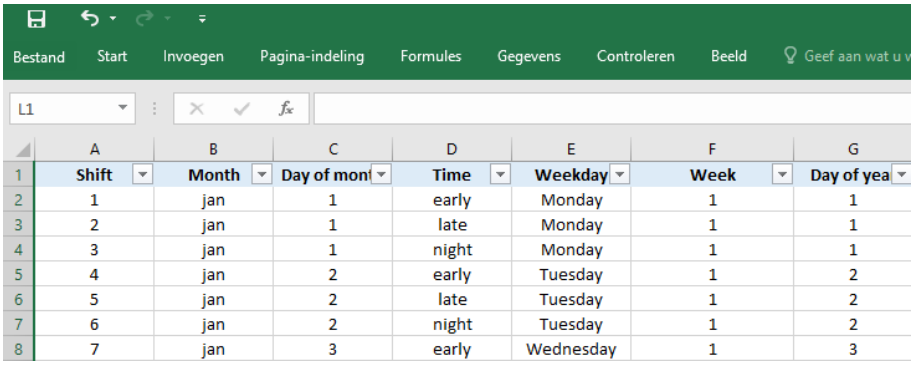
This section discusses the KB that contains the available information to create a good planning. A scheduling problem has some typical challenges. To start with, it works with time-dependent concepts. “day”, “week”, “month”, “next day” are very clear concepts in natural language, but they need to be defined for a computer system. That is the subject of the next paragraph. To make a distinction between must-be-met hard constraints and nice-to-be-met soft constraints, often an approach with a weighted cost function is used. This is the topic of the second paragraph. An important factor to keep in mind when designing scheduling systems is their time performance. Especially for requirements that necessitate a proportional distribution of something, rather than setting a maximum limit, there might be a considerable impact on the performance. This is explained in the third paragraph.

Concepts of time

We investigated three possible ways to manage dates in FO(·). The first possibility is to define date concepts in the KB itself, for example by numbering each shift, the early shifts are the shifts divisible by three, such that {! s[Shift]: GetTime(s)=early <- s%3=0.}. The advantage of this way of working is that it is relatively easy and quickly done, especially if the calendar year starts on a Monday. However, the disadvantages are that more time is needed to develop code for multiple years, it cannot be reused in other applications, and most important, the calculation need to happen with every run of the program, and this takes a lot of time. Another approach is to define the calendar in Excel, and read it as a structure by using a Python script (See Figure 8.2).

It is a quick and easy solution that has the advantage that the calculation does not need to happen in IDP. Moreover, the Python script can be used to input other parts of the structure as well, such as the availability of doctors. The disadvantage is that the calendar needs to be manually created in Excel for each year, and that the script needs to be amended for use in other applications. The latter is an important element, as dates and times are used in many applications. Therefore a separate date library that can be used by every IDP3 application was developed.[1] To use it, the library needs to be included in the IDP3 file, the vocabulary of the library needs to be included in the vocabulary of the calling specification. The imperative procedures from the library are called in the structure, that also contains a date range. Subsequently the vocabulary can

[1] bitbucket.org/krr/date-library-for-idp3



	A	B	C	D	E	F	G
1	Shift	Month	Day of mon	Time	Weekday	Week	Day of yea
2	1	jan	1	early	Monday	1	1
3	2	jan	1	late	Monday	1	1
4	3	jan	1	night	Monday	1	1
5	4	jan	2	early	Tuesday	1	2
6	5	jan	2	late	Tuesday	1	2
7	6	jan	2	night	Tuesday	1	2
8	7	jan	3	early	Wednesday	1	3

(a) Mapping between different concepts of the calendar in Excel

```
Structure S:V{
GetMonth={(1,jan);(2,jan);(3,jan);(4,jan);(5,jan);(6,jan);(7,jan);(8,jan);(9,jan);(10,jan);(11,jan);(12,jan)...}
GetDayOfMonth={(1,1);(2,1);(3,1);(4,2);(5,2);(6,2);(7,3);(8,3);(9,3);(10,4);(11,4);(12,4);(13,5);(14,5)...}
GetTime={(1,early);(2,late);(3,night);(4,early);(5,late);(6,night);(7,early);(8,late);(9,night);(10,early)...}
GetWeekday={(1,Monday);(2,Monday);(3,Monday);(4,Tuesday);(5,Tuesday);(6,Tuesday);(7,Wednesday);(8,Wednesday)...}
GetDay={(1,1);(2,1);(3,1);(4,2);(5,2);(6,2);(7,3);(8,3);(9,3);(10,4);(11,4);(12,4);(13,5);(14,5);(15,5)...}
GetWeek={(1,1);(2,1);(3,1);(4,1);(5,1);(6,1);(7,1);(8,1);(9,1);(10,1);(11,1);(12,1);(13,1);(14,1);(15,1)...}
}
```

(b) Resulting structure in IDP (parts of lists replaced by "...")

Figure 8.2: Parsing of Excel calendar in IDP structure

be used to create constraints in the theory. The library itself consists of Lua procedures to make the calculations, and a vocabulary.

Hard constraints and preferences

As mentioned before, this case only has a few hard constraint that need to be met in every schedule. The majority of constraints are preferences, meaning that in the ideal schedule they are met, but they can be violated if the combination of all constraints prevent the creation of a schedule. As different employees typically have opposing interests, it is typically very hard or impossible to take all preferences into account. The solution is to retain hard constraints as a constraint and to assign weights to the violation of preferences. In the end we will not be looking for a solution with the model expansion inference task, but we will look for a solution with the minimum associated weight (i.e., with the lowest amount associated to violations). In practice this means that our specification has the following structure:

```
vocabulary {
        ...
        Violation:(Doctor * Rule) → Bool
```

```
        Violation_Cost:(Doctor * Rule) → Cost
        ViolationPerDoctor:(Doctor) → Cost
        ViolationCostTotal:() → Cost
        ...
}
theory {
   ∀ d in Doctor : Violation(d,1) ⟺
        # {wd in Weekday, w in Week:Works(d,wd,NextW(w))} >5.
   ∀ d in Doctor:  ViolationPerDoctor(d) =
        sum{r in Rule:Violation(d,r):Violation_Cost(d,r)}.
    ViolationCostTotal() = sum{d in Doctor, r in Rule:
        Violation(d,r):Violation_Cost(d,r)}.
}

structure default{
        Violation_Cost :={(Dr.A,1) → 10,(Dr.B,1) →10,
        (Dr.C,1) →10, (Dr.D,1) →10,(Dr.E,1) →10,
        (Dr.F,1) →10,(Dr.G,1) →10, (Dr.A,2) →10,
        (Dr.B,2) →10,(Dr.C,2) →10, (Dr.D,2) →10,...}
}
```

For the explainability of violations, we created a violation rule for each specific constraint. This allows the straightforward detection of the rule that was violated. Alternatively, violations could be defined more generally, but this would nevertheless require different violation "levels":

- Violations related to a specific shift, e.g., assign no shift on Monday to doctor A if doctor A has worked during the weekend.
- Violations related to the number of shifts per week, e.g., a doctor that works part-time.
- Violations related to the number of shifts per month, e.g., limit the number of weekend shifts during the month.

In this framework different kind of preferences are modelled:

- Individual preferences versus collective preferences. An individual preference might be to work the early shift on Mondays. An example of a collective preference is that a doctor works the entire weekend, or is the entire weekend off.
- Absolute preferences versus relative preferences. A doctor can prefer to work two night shifts per month, regardless of the number of night shift of

other doctors. Because this is not related to other criteria (e.g., night shift of other doctors, the work regime of the doctor,..), we call this an absolute preference. A relative preference means that a criteria depends on the relative amount of shifts. For example, the number of shifts assigned to a doctor should be relative to his indicated work regime (full time, half time, one fifth), and should be fairly spread among doctors.

Some of the preferences, such as an equal distribution of violations, are very resource intensive to calculate. To each of the violations, a weight needs to be assigned. The higher the weight, the more important it is to respect this preference. The assignment of these weights is not an easy task: the importance given to preferences is often implicit, changeable and subject to bargaining (your preference is lower now because it was higher last time). Nevertheless, as these weights determine the final schedule, they are key to the acceptance of the schedule. In Section 8.5 we discuss how we solved this problem.

Computational performance

The creation of a schedule with the formulated constraints takes multiple hours. For a fully-automated planning this may not seem a problem (as this can run during the night), but this time is unacceptable if interactive changes are expected from the planner. Some of the formulas are clearly more demanding than others. For example a constraint that aims to distribute the number of shifts proportionally will demand much more resources than the constraint that each doctor can work maximum one shift per day.

8.5 Prototype

In this case the term *prototype* could cause confusion. On one hand we created multiple versions of a schedule that we used to elicit further requirements. These iteratively created new versions can be considered as prototypes of a schedule. We call this a schedule prototype. On the other hand, when we realised that it would be impossible to capture all requirements in a static system, we developed an interactive interface that would allow the planner to "play" with the planning until the planner considers it satisfactory. Here, the system itself is a prototype of the final application. We call this an application prototype.

8.5.1 Schedule prototype

In the context of requirements generation, the use of planning prototypes proved indispensable. During the first meeting, the planner had focussed mostly on individual preferences, such as which shifts can be assigned or not to which doctors. However, even without feedback of the planner, the first schedule prototype was clearly useless. We had only modelled the constraints given by the planner, and as a result the majority of shifts were occupied by three doctors. This led to the addition of "common sense" knowledge, such as the number of shifts and the number of night shifts that should be equally distributed between doctors, keeping in mind their respective work regimes. The second prototype looked much better from this point of view. It was sent to the planner for feedback. New requirements arose from this: if possible, the days off of a doctor should be grouped together such that the doctor can enjoy a delayed weekend, e.g., if the doctor worked in the weekend. This is called a stint (i.e., specific shifts that need to follow each other, in this case two days off after the weekend) [105]. Although the addition of this requirement resulted in an improved schedule, this still did not meet the requirements of the planner, who pointed out that some doctors are more flexible than others to change their preferences. Moreover, this flexibility is not static: a doctor might be very flexible during school months, but less during school closing. However, it is not possible for the planner to give a complete overview of this kind of information she uses to manually create a planning. We concluded that it would not be possible to create a schedule with the same quality as the manually created schedule, because of information that cannot be completely pinpointed by the planner.

8.5.2 Application prototype

Even though it seemed impossible to create an automatic schedule, this does not mean that it is not possible to develop an application that can support the planner in her planning process. To this end, we started from the generic Interactive Consultant. From the entire specification, we keep a limited number of symbols in the interface, subdivided in three classes: the planning class, the generic class and the rules class. The *planning* class contains the schedule of the doctors, i.e., the assignment of all shifts to a specific doctor as determined in the function $dec_Planning(Shift) : Doctor$. At the start of the problem, this is the decision variable of the application. The second block contains the *rules* and their associated weights, the total violation cost and the violation cost per doctor. This block allows the planner to assign heavier or lighter weights to a violation, or to determine that some doctor should have less violations,

or to disregard rules. The third block is the *generic* class. This contains a number of general inputs that are used further in the application, and that might change over time. However, the values in this block typically stay stable over a considerable amount of time. We assign values to these variables with a default structure. When using a default structure, their default value is assigned when opening the application. However, the user can manually change them if needed. It contains for example the work regime and availabilities of each doctor, as well as the impact of the work regime on the preferred number of shifts and weekendshifts. When a new schedule is created, assuming that it will not be immediately perfect, the planner can take three actions:

- Make amendments directly in the schedule. E.g., switch doctors of two shifts. If the planner manually amends the schedule, and wants to keep it as the final version, IDP should do a model check to see if the new model fits with the constraints (I.e., reset and choose some values of the planning, and reset Rules values, then expand model.
- Amend the constraints that have been violated. The variable *dec_planning* needs to be recalculated. IDP needs to do a new optimisation to calculate a new planning. In practice: reset and choose new values of rules heading, and reset planning heading. Then expand new model.
- A combination of the above, e.g., determine some values of the schedules and indicate some constraints to violate. IDP should then calculate a new schedule with this information.

To be able to do so, some values of the originally generated planning needs to be reset, while others need to be kept. We have developed reset functionalities at several levels:

- Two general reset buttons: one for values that have been set by the users, and one for choices of the system. The buttons will switch values set by the user (resp. system) to *unknown*.
- A heading reset button: all values for the concepts in this heading are switched to *unknown*.
- individual rest button: a specific value is switched to *unknown*.

The behaviour of the reset button depends on how the value was obtained. A value can be assigned in one of three ways: the user can assign a value, a value can be the result of the KB (e.g., by propagating consequences, or values in the structure), and the system can itself assign values by using the model expansion

inference task. If a value is reset, all of its logical consequences are reset as well. Figure 8.3 shows a part of the prototype screen with the different reset and propagation buttons.

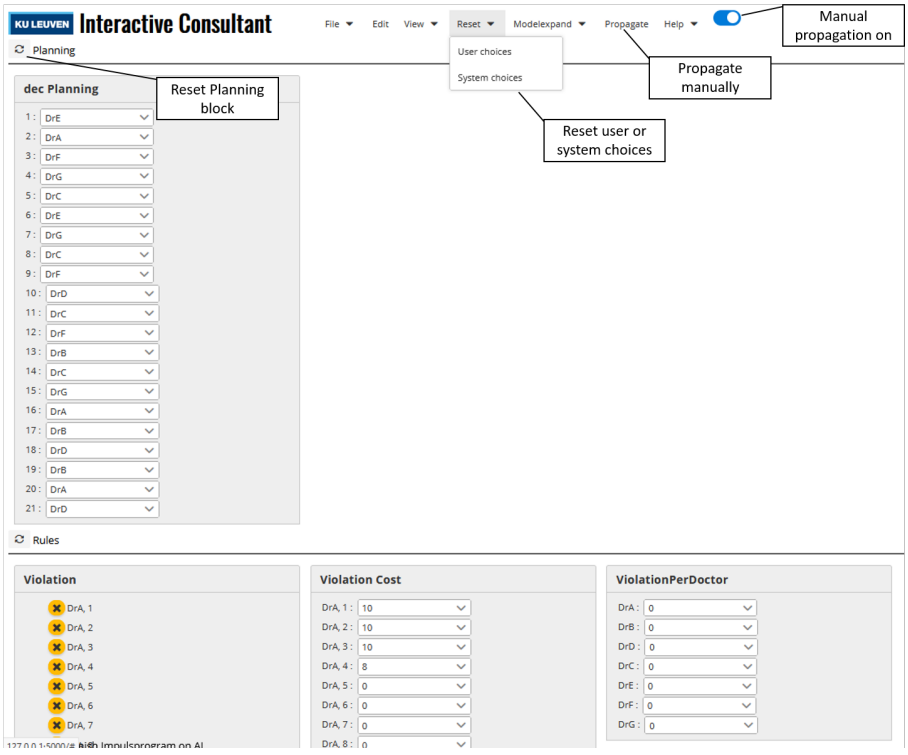


Figure 8.3: Adjusted IC for doctor scheduling with additional reset and propagation buttons

8.6 Conclusion

We have modelled the knowledge to create a doctor schedule for an emergency service in a small hospital. A strictly rule based modelling in DMN was not possible, as the scheduling problem is not a problem that can be hierarchically decomposed and described by deterministic rules. Alternatively, the constraint based cDMN notation did prove expressive enough to reflect the different constraints and preferences linked to the scheduling problem. The request for this case study came from the planner who currently creates the planning

manually. This person is not interested in the modelling of the problem, but only in the schedule that is created by the system. Therefore the modelling in the cDMN format has mostly as added value that it demonstrates the expressivity of cDMN in complex cases. The schedule prototypes that were created were very useful in prompting additional decision criteria that were not thought of before, such as a proportional distribution of the number of shifts and the grouping of days off. The final result however remained unsatisfactory for the planner. Therefore we proposed another approach. We accepted that we would not be able to create a fully satisfactory schedule due to the difficulty of assigning correct weights, flexible priorities of preferences, etc. We developed an interface in which a first planning is proposed to the planner. Afterwards the planner has the possibility to assign other doctors to shifts, adjust weights of preferences, to minimize violations for a specific doctor, etc. This allows them to optimise the schedule as preferred. The inference task of propagation, that is a standard part of the interactive consultant interface, ensures interactivity with the system because the consequences of each action is immediately shown. However, in this case the calculation of consequences after each adjustment may take a lot of time. Therefore we allow to turn off automatic propagation and replace it with manual propagation to calculate the consequences after a series of adjustments. This is an improvement, but it is insufficient for the planner to be able to play around with the application until a satisfactory schedule is obtained. Therefore we have not been able to test this solution in practice.

Chapter 9

Staff scheduling

In Chapter 8 we described the development and results from the case study on the creation of a doctor schedule. In this case study we encountered multiple difficulties, and the final result did not deliver an application that is used in practice. In this chapter we investigate the causes of this failure and formulate avenues for future research and formulate lessons learned for the development of staff scheduling applications. In the first section we describe important characteristics of scheduling applications, and how they are typically implemented. The second section zooms in on the kind of knowledge that is available for scheduling applications and how it can be obtained. This chapter concludes with recommendations to develop staff scheduling applications in Section 9.3.

Contributions A vast amount of literature exists on the creation of work schedules. In spite of this, the number of applications that result from it and are used in practice, are limited. Kellogg and Walczak [77] discovered in their literature review that only 30% of academic models is applied in practice. Van den Bergh et al. [133] found that from 196 published papers based on real-world data, only 46 (or about 23%) were implemented in practice. We contribute to the understanding by combining insights from operational research and social psychology. We draw on these fields to make recommendations for development of scheduling applications. Eventually we evaluate the suitability of the IDP system for scheduling applications against accepted quality criteria.

Defining a scheduling application A scheduling application is an application that supports the creation of a work schedule given a number of shifts to be filled and the available personnel. It is different from planning applications that aim to determine the number of shifts to be filled with personnel with a certain competence. We take the planning of shifts as a given for the development of a scheduling application. Our case study from Chapter 8 is an example of a non-cyclic, stable unit demand roster with counter, series and succession restrictions [124]. Non-cyclic means that rosters are not repeated in different periods. A cyclic schedule means that there exists only one (or a few) different rosters for all doctors, but that the place in the cycle is different for each doctor to cover the different shifts. E.g., imagine three shifts per day, to be filled by four doctors. This could be covered by one roster of an early shift, then a late shift, then a night shift, and a day off. However, at any given day, each of the four doctors is at a different place in the cycle, such that all shifts are occupied. This is illustrated in the schedule in Table 9.1.

	Monday	Tuesday	Wed.day	Thursday	Friday	Saturday	Sunday
Early	Dr. A	Dr. D	Dr. C	Dr. B	Dr. A	Dr. D	Dr. C
Late	Dr. B	Dr. A	Dr. D	Dr. C	Dr. B	Dr. A	Dr. D
Night	Dr. C	Dr. B	Dr. A	Dr. D	Dr. C	Dr. B	Dr. A

Table 9.1: Example of a simple cyclic schedule.

The demand is stable in the sense that the number of assigned doctors for each shift is the same on each day, more specifically, each shift is assigned one doctor, i.e., unit demand. Counter restrictions limit the number of a specific kind of shifts that can be assigned per time period to a doctor. The maximum number of weekend shifts is an example of this. Series restrictions limit the number of consecutive assignments. This can be a maximum consecutive assignments of night shifts, but also a minimum consecutive assignment of days-off. An example from our case is the requirement to group days-off in the week following weekend shifts. Succession restrictions are a specific kind of series restrictions. It puts a limit on two consecutive assignments. For example, one of the doctors that works on Sunday cannot be assigned on Monday. In our case, the doctors have the same qualification, so we do not need to take qualification into account to create a schedule.

9.1 Characteristics of scheduling applications

NP complete Many authors assume that scheduling is a NP complete problem. Smet has analysed scheduling problems in detail, and demonstrated that only

problems with a specific counter constraint can be solved in polynomial time [124]. Hence, the calculation of a schedule is a very resource intensive task. Only a reformulation of constraints without series requirements leads to a problem formulation that can be resolved in polynomial time.

The importance of preferences Also typical for scheduling applications is the number and diversity of rules and preferences that are involved. A planner not only has to take into account legal constraints, but also organisational requirements and individual contracts. Next to these binding agreements, each level also has its preferences, that possibly conflict with one another. In scheduling literature these preferences are typically expressed as weighted sum functions. Each preference receives a violation cost: a penalty assigned to the schedule if this preference is not respected. The aim is then to create a schedule with the lowest overall penalty. The experienced quality of the schedule hence depends (among other things) on a good assignment of weights. In Section 9.2.1 different ways to assign weights are compared.

Planning horizon Another difficulty to implement schedules in practice, is the continuity of planning. Typically, a planning is created for limited time period, such as a month. In practice, the planning of this month is preceded by the planning of last month, and will be followed by next month's planning. Manual rostering will take this in consideration, and ensure a consistent planning across periods. Special care must be given to automated planning to ensure consistency across subsequent planning periods.

Scheduler's job crafting activities The previous factors that might explain why automated schedules are not more often implemented in practice, relate to the quality of the delivered schedule. In this viewpoint, scheduling is a rational and straightforward process that follows some rules, and if we are able to capture those rules well enough in our automated schedule, it should be accepted. This point of view ignores the specific role of personnel planning in companies, and especially in the health care sector that has 24/7 shift coverage. Several studies show severe consequences of poor nurse staffing decisions, not only for the nurses themselves (in the form of job satisfaction, mental health and safety), but also for patient care and even patient mortality [5, 80]. Together with the fact that health care professionals are bottleneck profession [140, 73], it becomes apprehensible why taking the preferences of employees into account is important. Schedulers therefore occupy a central, and often powerful, role in many health care institutions. To meet the various expectations (legal, organizational, personnel), schedulers engage in job crafting

activities (i.e., physical and cognitive changes schedulers make in the task or relational boundaries of their work) [80]. Examples of such scheduling strategies are asking employees to fulfil a request in return for a favour, construct rules of thumb to ensure fairness of the schedule and executing additional tasks that surpass the actual scheduling task. Therefore, scheduling is more than an organizational role, it is a social phenomenon in the workplace [80]. Hence, replacing this with an automated schedule might not be possible or even wanted. Note that also rule enforcement, a strategy in which the scheduler sticks rigidly to the scheduling policy of the company, is considered a job crafting strategy. Where planners use this as their predominant strategy, the acceptance of automated schedules might be higher.

9.1.1 Creating a qualitative roster

When studying rostering problems, the question arises what a good roster looks like. This seems to be very subjective, as preferences are different from one person to the other. Fairness is an important quality of personnel rosters [109], but this shifts the question to what a fair schedule is. Petrovic and Van de Bergh [110] propose seven criteria to compare not the schedules themselves, but the approaches to come to a schedule. These are:

- Expressive power to represent the different constraints present in personnel rostering.
- Flexibility to add or remove constraints.
- Algorithmic power to calculate a good solution in limited time.
- Learning capabilities of the system to improve its performance over time.
- Maintainability of the system: how easy is it to update knowledge?
- Re-scheduling capabilities to quickly offer a new qualitative and fair roster
- Parameter tuning such that they are convenient for a wide range of instances.

Fairness in Personnel Rostering

As mentioned before, most personnel rostering models use a combination of hard constraints and soft preferences, that receive a weight that adds to an objective function each time the preference is violated. The result of such a

model can be considered fair if the violation costs are balanced over the different employees [79]. Multiple fairness measures have been proposed, based on the difference between the maximum and minimum violations over employees [126], the standard deviation of individual penalties [26], or Jain's fairness index [95]. Komarudin et al. propose a combination of three criteria to examine the fairness of a roster: i) the total violation cost, ii) the standard deviation of individual violation costs, and iii) the maximum individual violation cost [79]. As the fairness objective may be contradictory to the roster quality objective, both objectives are combined in a heuristic that focuses first solely on the total violation cost, and only uses fairness criteria in the second phase [79]. Another approach is the use of a satisfaction indicator, as proposed by Gross et.al. [59]. The idea is that traditional criteria are linked to the planning horizon. Hence, relatively fair schedules for a given month, may result in an unbalanced long-term overview. The satisfaction indicator takes the preference fulfilment of previous months into account to calculate a new schedule [59]

9.1.2 Quality of IDP to create personnel rosters

The quality criteria to evaluate scheduling systems are now used to evaluate the quality of IDP as scheduling system. In this section we evaluate the combination of the FO(·) KB language with the IDP reasoning engine and the interactive consultant interface.

Expressivity: present The FO(·) language proved to be expressive enough to formulate the knowledge required to create a schedule. This includes the distinction between hard and soft constraints, and the creation of a cost function, working with time concepts (e.g., NextDay), composed concepts (e.g., GetDay(NextShift(s))), comparisons, etc.

Flexibility: present This criterion points to the flexibility to add or remove constraints when requirements change. In FO(·) each sentence is independent of the other. This modularity makes it possible to add or remove sentences. It is also this quality that makes it fit as evolutionary prototype that can start very simple but can be developed into a complex system.

Algorithmic power: to improve The algorithmic power ensures the calculation of a schedule in limited time. Although small instances with a limited set of rules are calculated almost instantly, performance quickly decreases when larger instances or more rules are used. This is true for both the IDP3 system with

underlying Minisat solver, as for the IDP-Z3 system with underlying Z3 solver. Especially for an interactive system that allows the planner to modify parts of a first planning, the calculation times are unacceptable.

Learning capabilities: integration with other systems The IDP system itself does not contain learning functionalities. In other cases we demonstrated that the IDP system can be integrated with other systems, such as a natural language module (Chapter 10) or a historical design database (Chapter 6). An integration with a learning module and development of learning pipeline can be developed.

Maintainability: present The maintainability of the system means that knowledge in the system can easily be updated. We discussed previously how constraints can easily be added or removed. For the same reason it is also easy to update existing constraints. The constraint in question can easily be identified, and adapted without affecting other constraints.

Rescheduling capabilities: present As stated before, the FO($\cdot$) language offers expressivity to formulate various constraints. This includes the formulation of constraints on the new schedule in case of rescheduling. Typically a new schedule should contain as little changes as possible. The only limiting factor is the algorithmic performance of the system, that hampers the quick creation of a new schedule.

Parameter tuning: present In FO($\cdot$) there is no distinction between input, intermediate and output variables. Hence, parameters of the model can also be treated as inputs in the interactive consultant. The planner is able to adjust them as needed.

We conclude that the IDP system has some promising characteristics for the creation of qualitative schedules. However, the limiting factors (restrained algorithmic power and the absence of learning capabilities) need to be addressed before become a viable solution.

9.2 Knowledge in scheduling applications

As mentioned before, due to the different concerns to take into account, scheduling is a complex task. Schedulers have different kinds of knowledge they use to complete this task.

Domain knowledge To start with, for a scheduling application the planner knows the required resources, and the problem is limited to assigning the correct number of employees to every time slot. The determination of these required resources happens earlier and is called a planning or staffing problem. The domain knowledge of schedulers encompasses all the rules and preferences of the different stakeholders in the scheduling process: legal obligations, organizational guidelines, contractual obligations, preferences of employees, etc. Some of the knowledge is hard to access for planners. For example, if asked to assign weights to different preferences of employees, this might be a time consuming or even impossible task. The planner typically also knows how these different concerns were previously implemented in the company, and sometimes keeps metrics of specific schedules, e.g., the number of worked holidays. If the planner engages in job crafting activities, they probably wield additional rules and heuristics that can be considered a form of domain knowledge. Besides the expert knowledge from the planner, other sources can also be consulted, such as labour law on work and rest times and sectoral agreements. Institution procedures and labour contracts are another source of information. These documents typically contain information on accepted schedules (i.e., the hard constraints). The main source of soft constraints are the employees themselves, on which the planner has only a partial view.

Task knowledge The task knowledge refers to the steps that the planner takes to create a schedule. This can be different for each planner. One possibility is that the planner has some kind of "template schedule" in their head, and that they start from this template and amend it locally to arrive at a specific schedule for the period. Another possibility is that the planner starts with planning difficult shifts, typically weekends and nights, and fills in the others afterwards. The activities performed by the planner can differ considerably, and can be related to the way the planner carries out their function. For example, a planner that sticks to a rule-bound interpretation of their job, will rely on by-the-book policy enforcement and restrict personnel flexibility. On the other hand, planners that engage in relational job crafting might bargain with employees to take certain shifts, or take actions outside their function to find a solution for an employee [80].

Inference knowledge The different inferences used during automated scheduling are propagation, model expansion and optimisation. For manual planners these different inferences will be translated in specific actions to take during the planning process, but this is likely to be transparent for them.

9.2.1 Creating fair schedules

As discussed previously, the topic of fairness is important in scheduling applications. Hence, multiple papers have been devoted to the subject. These papers consist of algorithms to construct a fair schedule, however without diving into the concept of fairness. This is however a multifaceted concept, and should not be confused with the concept of equity [89]. Thus more effort should be devoted to understand it in depth. As mentioned earlier on, one of the job crafting activities that might explain the difficulty to develop automated scheduling applications, is the creation of fairness measures by the planner. Therefore, to be able to develop an application, we should try to understand what this means in abstraction of concrete situations. Fairness is often said to be in the eye of the beholder [16], so depending of the point of view. Then how can we derive general guidelines? This can be done by making a distinction between the fairness of a schedule, and the fairness of the procedure by which the schedule is created. In social justice literature, these forms of fairness are called distributive fairness and procedural fairness respectively [89]. The good news from a development point of view, is that generally the procedural fairness is valued higher than the distributive fairness [16], because it means that for a creation of a fair schedule not every single schedule needs to be scrutinized, but that it suffices to agree on a fair procedure that can subsequently be used to create all schedules. Already in 1980 Leventhal distinguished six rules of procedural fairness [89]. The first one is the *consistency rule*: an allocative procedure should be applied consistent across persons, time and situations. The procedure itself should be balanced and free of self-interest and narrow preconceptions (i.e., the *bias-suppression rule*). The procedure should take into account as much as possible aspects that influence the decision, which is called the *accuracy rule*. Moreover, the views and basic concerns of all stakeholders should be represented (the *representativeness rule*). Of course, the procedure should also be morally acceptable (i.e., the *ethicality rule*). Finally, the *correctability rule* says that it must be possible to correct the decision in case an error has occurred.

Creating a fair scheduling procedures

The rules formulated by Leventhal [89] can be translated to the context of personnel scheduling.

- Consistency: as the scheduling procedure should remain consistent for at least some time, it is a good idea to take less common circumstances into account at the outset of the project. For example, preferences might

change during holiday periods, so it might be a good idea to develop a system that allows this shift in preferences for the period, such that the scheduling procedure itself can stay the same. If changes to the procedure are required, it would be good to have a separate procedure in place for changing the scheduling procedure.

- Balanced procedure: It is self-evident that the planner should not grant themselves favours if they are part of the schedule they create. A bigger challenge might be to keep an open mind with regards to preferences. For example, the planner might assume that employees prefer to perform as little night shifts or overtime as possible. But some employees might prefer these shifts for their higher compensation or to combine their job with their personal situation.
- Accuracy: Try to gather as many elements that can impact the schedule as possible and confirm with various stakeholders that you have covered them.
- Representativeness: If the purpose of the exercise is not only to duplicate the schedule that is made by the planner, but to deliver more qualitative schedules that will be used in practice, the different stakeholders should be involved in the procedure creation process.
- Ethical: the legal and contractual rules will be implemented as hard constraints in the schedule model. Also additional ways of working that might exist, such as quid-pro-quo arrangements, should be ethical.
- Correctable: the application should offer the possibility to generate a new planning if one of the inputs was erroneous, with as little impact on the originally generated planning possible.

Assigning weights

As stated before, many scheduling algorithms assign weights to the different soft constraints with the purpose of minimizing the violations of these constraints. The question of how to assign these weights, is less well described. The procedure on how to do this, can be part of a fair procedure described above. Smet [124] proposes to derive the relative importance of preferences from past schedules. For each constraint, it is counted how many times the constraint is fulfilled, and how many times it is violated. The weight that is assigned to the constraint is the proportion of respected constraints in the total instances of the constraint, or formally:

$$weight_c = \frac{\#respected_c}{\#respected_c + \#violated_c}$$

with $weight_c$ the derived weight of constraint c, $\#respected_c$ the number of times the constraint was respected and $\#violated_c$ the number of times the constraint was violated.

This method is simple, and yet effective to assign weights to constraints. However, it does have some disadvantages. To start with, as it is based on past schedules, the new schedules based on these weights can only hope to deliver the same quality. The new algorithm will not create better schedules. It also means that enough schedules of good quality need to be available. If the circumstances change (e.g., new departmental regulations, or new doctors with new preferences join the team), (a part of) the model needs to be retrained and new training data might not immediately be available. Moreover, weights may not be static. For example, a doctor's preference to not work on a specific day might be very important during the school year due to the doctor's child care organisation, but it might be less important during holiday periods. A planner will probably take this into account when creating a manual planning. Therefore it may be preferable to assign weights manually. This is however a difficult exercise, and following factors should be taken into account:

- Frequency of possible violations: constraints can be expressed over different time periods: per day, per week, per month,... If the same weight is assigned to a day constraint as to a month constraint, the first one will have a bigger impact on the planning, as it occurs more frequently.
- Change of preferences: preferences may not be fixed, in the sense that their relative importance may vary throughout the year.
- Quantifying preferences: it is typically very difficult to express preferences in numbers. On the other hand, the interpretation of a number is not straightforward. People find it often easier to order preferences without giving a numerical interpretation.
- Who decides? Is the planner the one who will assign weights to the various constraints, will employees be asked to assign weights, or a combination of both?
- Degrees of freedom: the assignment of weights can be limited to promote fairness. In the most liberal situation, weights can be assigned independently for each constraint and each individual. This freedom can be limited in several ways. It can be decided to assign the same weight for all employees. Or the same weight can be assigned to general constraints, while differences are allowed for individual constraints. Or a total weight can be granted to each employee, who is free to distribute it over the different constraints as preferred. This can be limited again, e.g., by allowing maximum half of the granted number to one constraint.

- Are preferences of all employees treated equally, or are the preferences of some employees considered more important? For example, seniority could be a factor to defend a higher priority.
- What are the trade-offs between preferences? Many approaches use a weighted cost function to deal with the multi-objective nature of rostering problems. However, in practice there exist trade-offs between preferences that might impact the acceptable solution. In this case another approach, such as the use of thresholds for compromise solutions might be a better way to deal with preferences [15].

In our application we have chosen to not determine weights, but add them as a parameter to the interface. This allows the planner to change them to obtain a feasible planning. Although this might be something that currently happens subconsciously when creating manual schedules, this way of working may be hard to defend from a fairness point of view.

9.3 Recommendations

Given the difficulties related to the acceptance of scheduling applications, the knowledge engineer should be perceptive to factors that impact it. By using structured interviews as format, both job crafting activities by the planner and constraints that impact the schedule can be probed for. Kossek et al. [80] used an interview protocol for their research on job crafting activities. We propose to use a slightly amended version to put more focus on the actual constraints and preferences to be modelled (see Appendix A). The result of the interview can be processed in three ways: i) extract constraints and preferences to create a planning and model them in the KB, ii) search for clues of job crafting activities. This can be done by analysing the answers from the planner, and tallying whenever an activity of the first-order codes of Kossek et al. is mentioned. This reveals the axial codes and eventually job crafting activities, as demonstrated in Figure 9.1 [80]. For example, when a planner says that priority is given to doctors with higher seniority, this is a statement of being fair (first order code), which indicates that the planner develops constructions of fairness (axial code), and hence engages in cognitive job crafting (job crafting activity). iii) Determine the purpose of the application. Largely speaking the purpose can be to have a similar schedule as before, but in limited time, or it can be to have an improved schedule.

If the purpose of the activity is to automatically create similar schedules as before, and the planner does not (or very little) engage in job crafting activities,

FIRST ORDER CODES	AXIAL CODES	THEMATIC CODES
- Examples of putting one stakeholder group before another - Statements about needing to balance multiple stakeholder needs	Stakeholder Prioritization	Cognitive Job Crafting
- Statements about being fair - Statements about treating people equally	Constructions of Fairness	
- Descriptions of tasks performed above and beyond formally defined role	Role Extension	Physical Job Crafting
- Descriptions of development of new policies and practices to aid in scheduling	Innovative Practices	
- Statements involving trading favors with employees - Examples in which schedulers or employees ask for something in return for fulfilling a request	Quid pro Quo	Relational Job Crafting
- Examples of empathizing or sympathizing with employees - Examples of recognizing employee problems or needs	Emotional Support	
- Descriptions of helping employees get their preferred schedule - Examples of going above and beyond to help employees get schedules that fit their needs	Instrumental Support	
- Examples of refusing to make exceptions to scheduling policies - Examples of refusing to adapt policy and practice to meet individual employee needs	By-the-Book Policy Enforcement	Rule-Bound Interpretation
- Examples of refusing to make changes in the schedule - Examples of employee's requests for time off being denied	Restricting Flexibility	

Figure 9.1: Determining job crafting activities based on first order codes by Kossek et al. [80].

or they engage in rule-bound interpretation, it might be relatively easy to model constraints and preferences. Weights can be assigned according to the historical schedule method proposed by Smet [124]. If the planner refers to role extensions, such as developing constructions of fairness, this is additional information that should be exploited further in a next interview.

We assume when planners engage heavily in job crafting activities, it becomes less evident to automatically create a schedule. These kind of job crafting activities are difficult to elicit by simple interviews. The use of a prototype planning can reveal detailed examples. An interactive system that supports the planning process, as we developed, may be a solution. In this case, prototyping can be used to probe for the task and inference knowledge that the system must be able to cope with.

If the purpose is to improve the schedule, another possibility is to redesign the planning process, such that all job crafting activities are exposed and evaluated if they fit in a fair scheduling process. This kind of project would not only

involve the planner, but also other stakeholders, such as (representatives of) employees. It has a large change management component and falls outside the scope of this thesis. Scheduling is both an important and complex task, and so is the creation of an application for automatic scheduling. Before launching a project, one should be aware of the obstacles that might hinder the actual acceptance in practice. These obstacles can be grouped in three themes:

- Performance challenges: the calculation of a schedule is very resource intensive
- Quality of the system and of the resulting planning, e.g., discrete planning periods in the system versus continuous planning in reality.
- Planners occupy a social role in the organization, and are very likely to engage in job crafting activities.

Therefore at the outset of the project, the question should be whether a fully automated planning is intended, or if an interactive support is preferred. To create a fully automated planning, the common way of working is to distinguish between hard and soft constraints. A penalty is assigned to the soft constraints, and each time a planning violates the soft constraint, the penalty is added to the overall violation cost. An optimization inference calculates the schedule with the lowest associated violation cost. The choice for an interactive system is preferable if no solution can be found based on the hard constraints, if it is not possible to (meaningfully/fairly/...) assign weights to identified soft constraints, or if it is not possible to formulate all constraints. Fairness is an important characteristic of qualitative schedules. One way to look at it, is to balance certain metrics, such as the number of night shifts. This is a distributive view on fairness, i.e., only the result is taken into account. When the workforce is very diverse a procedural view on fairness may increase employees opinion of the schedule. In this case the fairness of the schedule is not evaluated on a case-by-case basis (i.e., by evaluating this months' schedule), but by setting up a fair procedure to create the schedule. The involvement of different stakeholders to create a balanced procedure, and the consistent application of this procedure are important elements that contribute to fairness.

Chapter 10

Cases: Third party applications

In this chapter we present three cases that are very diverse in multiple viewpoints (different industries, different company profiles, etc.), but that have one characteristic in common. All of these applications have been developed for companies that have in the end not the entire knowledge to build the KB. A more or less substantial part of the knowledge is only possessed by an external party. Whereas typical application development necessitates interaction between two parties – the development team and the business users – the external client is a third party involved in the development. Therefore we call this third party applications.

The first case (Section 10.1) is the creation of an application that is used by an international financial institution to facilitate the collateral management of their clients. The second case (Section 10.2) entails the development of a system that an international vendor of railway communication systems could offer to their clients to tailor on-board communication. The third case (Section 10.3) describes a system to process working time registrations according to existing legislation on the one hand, and company specific guidelines on the other hand. As a part of the knowledge is not accessible, the analysis from a knowledge point of view is less thorough than the analysis of the other cases. Also the knowledge activities are less comprehensive. Therefore the generalisation is more concise, and will be presented at the end of this chapter (Sections 10.4 through 10.6).

10.1 Case 1: Collateral profile management

In this section we present an application that we developed to assist users with the creation of a collateral profile for the selection of financial assets. It consists of a natural language (NL) interface, an automatic translation to a declarative FO(·) KB, and the IDP reasoning engine with multiple forms of logical inference. The application was developed in collaboration with Coppens and Partners, who developed the natural language interface and the automatic translation of NL sentences to FO(·). Simon Vandevelde and Jo Devriendt assisted with the implementation. The results were presented at the Fourth Women in Logic Conference [1] and at 18th International Conference on Principles of Knowledge Representation and Reasoning (KR2021) [2].

Contributions In this case we developed a FO(·) model of the problem domain such that it can be integrated with a natural language module. We analysed the required services and decomposed them in subtasks with their associated inference tasks. New inferences to compare rulesets and create a reproduction in a legal *Appendix A* were developed.

10.1.1 Introduction

Collateral is a financial product that has the purpose of mitigating the non-repayment risk of interbank loans. It consists of financial assets that are considered robust enough to keep their value if the lender needs to market them. The eligibility of a specific asset depends on a plethora of interfering rules and constraints that are agreed upon in a collateral agreement between collateral giver (the borrower) and collateral taker (the lender). To automate collateral selection, these rules and constraints are typically formulated in lengthy programs that contain a lot of enumerations, repetitions, and complex nesting of if-then clauses and exceptions that need to be followed in the right order. This makes each creation of a collateral selection program a complex and time consuming task, and the result is hard to validate. For the management and exchange of the collateral, often an intermediate third party is used. Typically, it receives a description of the collateral profile in NL instead of the collateral agreement that has been signed between the parties. We have developed an application on behalf of such an intermediate third party to support their clients to optimize their collateral process.

10.1.2 Requirements

The first concern of the company is to improve the processes of collateral profile creation and effective collateral selection. In the previous way of working the collateral giver sent a description in NL to the treasury operations department of the company. An operator interpreted the profile, and created a so-called *Appendix A* based on it. This Appendix A needs to be validated by the client, and is a part of the legal contract between the company and its client. It is also the direct input of the system that selects the collateral. It contains the collateral profile of the client in an ordered list of rules that are processed top-down. The process of creating and validating the Appendix A easily takes months, and the outcome is not always what the client actually meant because it is hard to interpret. The company wants to shorten the process, make it better understandable for their clients, and deliver additional services to allow clients to better manage their collateral. Another important requirement is the correctness of the system. In the international financial market immense amounts circulate, and a short position of only one night involves considerable interest payments and may damage confidence in the defaulting party. If a technical problem at the third party service provider caused it, costs will be reclaimed from the service provider and may have severe repercussions on its image.

10.1.3 A natural language interface

To enable clients to manage their own collateral profile, we propose a declarative reasoning system with a NL interface. It allows the eligibility of financial products to be defined by means of controlled natural language (CNL). Each sentence is constructed from a number of building blocks that are selected step by step to get to a complete sentence. The resulting highly structured NL sentence is automatically translated to FO($\cdot$). For example, the first sentence from Fig.10.1 results in the following FO($\cdot$) sentence:

$$\{\forall a[Asset] : Eligible(a) \leftarrow AssetType(a) = bond \wedge Outstanding(a) > 1000 \wedge$$

$$Outstanding(a) < 2000 \wedge Industry(a)\ in\ (Automotive, Banking).\}$$

The building blocks are defined in a manually created *tuple tree*. The standard tree in our application consists of 400 nodes and took two weeks to construct. Each node represents a concept, like country or asset type, and each concept accepts possible values related to it. Users can extend the available picklist of values by introducing and defining new concepts. E.g., in the standard tuple

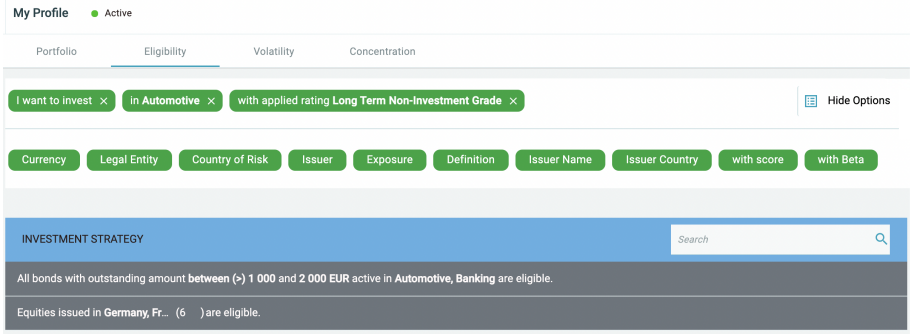


Figure 10.1: Creating a CNL sentence with building blocks.

tree, "rating" can take different values, from AAA, AA, A,... to D. A user could create a definition "Investment Grade", defining it as ratings above BBB. "Investment Grade" then becomes a new value in the picklist, that can be used to construct CNL sentences.

In addition to the interface to construct CNL sentences, the application also contains a deep learning natural language processing (NLP) module that accepts free-form English. It then proposes three CNL statements that are most likely to present the English sentence (see Figure 10.2). The user then selects the most correct sentence, makes adjustments if necessary, and validates the result. As before, this CNL statement is added to the KB in FO(·). The NLP module consists of a custom attention-based network, that was implemented in Tensorflow. The models use a sequence-to-sequence architecture. The training data uses a combination of real and synthetic data. The real data are previous interactions from the users. The synthetic data is achieved by creating random walks on the tuple tree and several grammatical transformations to achieve a rich set of examples. The model is trained with an Nvidia RTX 2080 with training times from three to five hours.

The possibility to enter sentences in (controlled) natural language allows the users to create a complete and correct KB of their investment profile.

10.1.4 The knowledge base system

When completed, the KB that was obtained by translating the user-entered NL sentences can be used by different inference methods to perform multiple tasks in the problem domain. This section discusses the underlying IDP reasoning

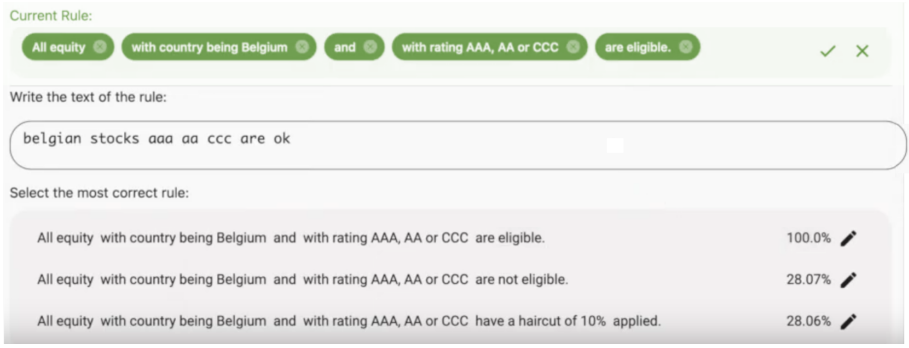


Figure 10.2: Natural language statement and its interpretations.

engine of the application, which consists of a KB at one hand, and inference tasks that can be applied to the KB on the other hand.

Structure of the KB

In the application two KBs were integrated: one to determine the eligibility of assets for collateral, and one to maintain concentration limits.

Deciding on the eligibility The KB is constructed by translating the CNL sentences to FO($\cdot$). The CNL sentences can end in one of two possible ways: either the asset is *eligible,* or it is *not eligible.* Every sentence is created independently of other sentences. This means that for a given asset, contradicting rules might occur. In this case, the non-eligibility rule should take precedence over the eligibility rule. To reflect this in our theory, we create two separate predicates *'Eligible(asset)*' and *'NotEligible(asset)'*. It is the predicate *'Eligible(asset)'* that determines if an asset is eligible for selection in the profile. The theory consists of two definitions, one for each predicate.

A definition in FO($\cdot$) consists of a set of rules of the form $head \leftarrow body$. In our setting, all rules in the same definition have the same head. Each body is a sufficient condition for the general head to be true. Together, the rules are also necessary, i.e., the head can only be true if at least one body is [36]. The *NotEligible(asset)* is used as an exception in the definition of *Eligible(asset)*. The general structure of our theory is shown in the example below:

$$\left\{ \begin{array}{l} \forall\ a[Asset] : Eligible(a) \leftarrow \varphi_1\ \&\ \neg NotEligible(a). \\ \forall\ a[Asset] : Eligible(a) \leftarrow \varphi_2\ \&\ \neg NotEligible(a). \\ \forall\ a[Asset] : Eligible(a) \leftarrow \varphi_3\ \&\ \neg NotEligible(a). \end{array} \right\}$$

$$\left\{ \forall\ a[Asset] : NotEligible(a) \leftarrow \varphi_4. \quad \right\}$$

These two definitions are equivalent to the formulas:

$$\forall\ a[Asset] : NotEligible(a) \Leftrightarrow \varphi_4.$$

$$\forall\ a[Asset] : Eligible(a) \Leftrightarrow \neg NotEligible(a)\ \ \wedge (\varphi_1 \vee \varphi_2\ \ \vee\ \ \varphi_3).$$

Maintaining concentration limits Typically, an investor will not only describe the kind of assets they want to accept in their investment portfolio, but they will also have requirements for the profile of his entire portfolio. A sound portfolio should be diverse (e.g., in terms of asset countries or industries) and investors implement this by defining *concentration limits*. For the enforcement of these concentration limits we create a separate theory, as it concerns the next step after determining the eligibility, i.e., the actual selection of eligible assets. An example to force concentration limits on countries is shown below.

We start by calculating the concentration amount of each country, that is the sum of the market values of each asset with this country:

$$\forall c :\ CA(c) = sum\{a\ v : Country(a) = c\ \wedge\ MV(a) = v : v\}. \tag{10.1}$$

with *CA(c)* the concentration amount of country c, c the *Country* of asset a, and v the market value *(MV)* of asset a. The *sum*-aggregate consists of three components: the variables a and v over which it ranges, a condition that a and v must satisfy $(Country(a) = c\ \wedge\ MV(a) = v)$, and the value that needs to be summed up (v). The concentration, expressed as percentage, is given by

$$\forall\ c :\ CONC(c)\ =\ (CA(c) \times 100/TMV). \tag{10.2}$$

with *CONC(c)* the concentration percentage of country c and TMV the total market value of the portfolio selected for collateral. In other words, the TMV represents the collateral amount that needs to be pledged to the collateral taker.

Finally we force the concentration limit by comparing the user-set concentration limit percentage l of each country c with the concentration percentage in the portfolio:

$$\forall\ c\ l :\ CL(c, l) \geq\ CON(c)\ . \tag{10.3}$$

with l the concentration limit CL set by the user for country c.

Supporting clients to maintain their concentration limits In the specification described above, the system will only be able to tell if the concentration limit can be respected or not. In the latter case, no model will be returned, and the system will display an unsatisfiability message. As concentration limits are multi-dimensional (e.g., combining industry, country, currency, maturity date, etc.), it can be troublesome to detect which asset can be bought without breaching an other concentration limit. To assist clients in this process, we developed a specification that allows the system to calculate which kind of assets can be safely bought without jeopardizing any of the concentration limits. The purpose is not to advice the purchase of a specific asset, but rather to advice on the characteristics of the asset to buy. To this end we create dummy assets. The dummy assets reflect baskets of securities (e.g., for gilts, issuer country = UK, currency = GBP and issuer = UK government are given). The amount of the dummy asset is left open to be determined by the system.

Haircut The market value of the collateral to be paid to the counterparty does not exactly equal the needed collateral amount. To protect the receiver from sudden movements in the market value, a margin is added. This margin is called the haircut that is expressed as a percentage of the market value, and depends on the kind of asset. Typically the number of parameters taken into account to determine the haircut is limited to three to five parameters. It is important that each asset receives a haircut percentage. By default it is set at 100%, meaning that the entire value of the asset is counted as haircut, or in other words that the asset is not eligible. The formulation in NL sentences make them well-understandable, but does not promote consistency and completeness checking. These are characteristics of decision tables. Therefore the application visualises the haircuts in tables (Figure 10.3).

Inference tasks

The information that is declaratively stated in the KB, can be used for different purposes. This section describe how different inference tasks can be applied to implement different requirements.

Several of these inference tasks take as input not just the KB T itself, but also a partial interpretation I_p for part of the vocabulary V of T.

Model expansion The inference task of *model expansion* can be used to decide on the eligibility of a specific asset. Consider the following highly simplified theory. If an asset has a triple A rating, it is investment grade (IG). If the asset is IG and is issued in Belgium (BE), it is eligible:

$$\{\forall\ a[Asset] : IG(a) \leftarrow Rating(a) = AAA.\}$$

$$\{\forall\ a[Asset] : Eligible(a) \leftarrow IG(a) \wedge Country(a) = BE.\}$$

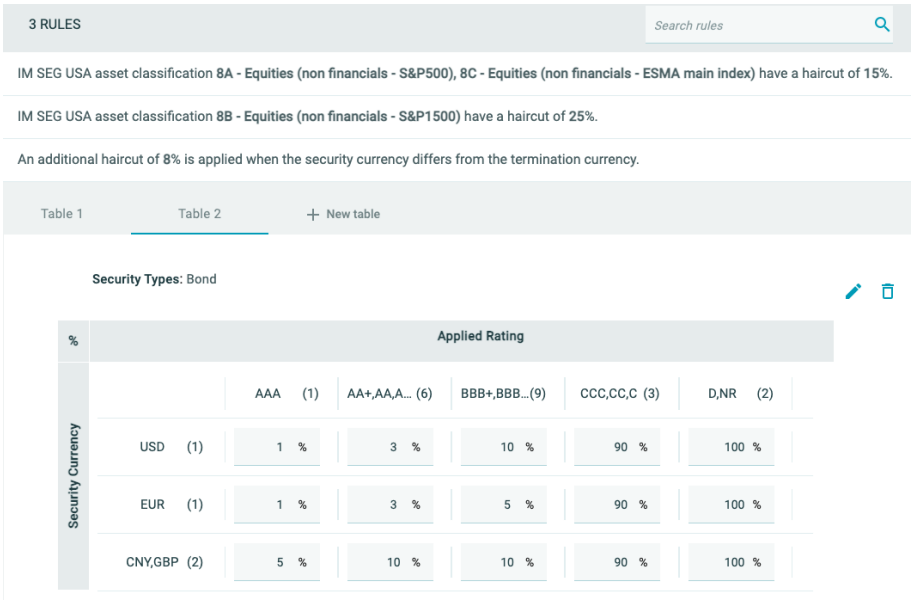


Figure 10.3: Illustration of haircut rules in a table.

The interpretation I_p that contains $Rating(asset) = AAA$, has two model expansions I_t:

Rating(asset)=AAA	Rating(asset)=AAA
IG(asset)=true	IG(asset)=true
Country(asset)=BE	Country(asset)=FR
Eligible(asset)=true	Eligible(asset)=false

In our application we typically possess all the information on the asset, such that only the values of *Eligible* and *NotEligible* need to be computed.

Optimization The *optimize* inference can be used to find a combination of eligible assets on which the lowest possible haircut is applicable. To this end we create an additional term m that represents the haircut dollar amount. This is, it will select the combination of assets with the lowest associated haircut amount that still follows the eligibility rules and given I_p. This inference task can also be used to give clients advise about additional assets to purchase in order to be able to respect concentration limits. As described in the previous section, in paragraph "Supporting clients to maintain their concentration limits", in case the existing portfolio of a client does not hold the

appropriate assets to respect the concentration limits, we create dummy baskets that indicate which kind of assets (i.e., with which characteristics) the client can purchase. It is the idea that as little possible additional assets should be obtained. Therefore a term n that contains the $Total()$ amount will be created to minimize the total position, and hence the number of assets to be purchased.

Propagation In the example above, if the partial interpretation I_p contains $Rating(asset) = AAA$, then the value $Investment(asset) = true$ is propagated, since it will be true in every possible model. Nothing can be derived about the value of $Eligible(asset)$ because this still depends on the country. In the application, the propagation works interactively: as soon as one value is selected, or a new rule is created, the impact on the eligibility is immediately shown by coloring the asset green (eligible) or red (not eligible) as shown in Figure 10.4.

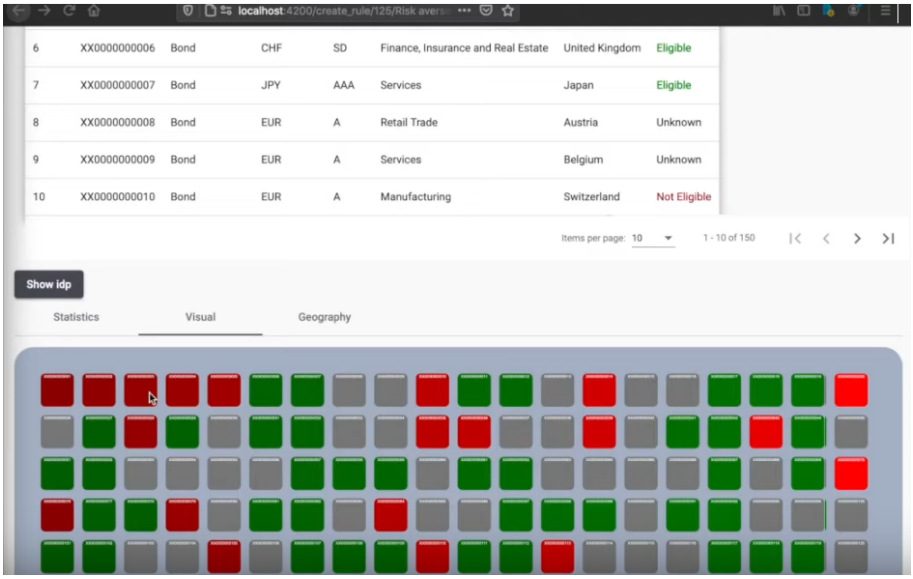


Figure 10.4: Visualization of interactive propagation.

Explanation The *explanation* inference traces the propagated values back to the given values of the interpretation I_p [3]. The application allows the user to click on a propagated value and see immediately which atoms steered the decision. This helps the user to understand unexpected propagations, for example in case an asset is not eligible, when the client expected that it would be. In the running example, the explanation of $InvestmentGrade(asset) = true$ is that $Rating(asset) = AAA$.

Explain unsat In Section 10.1.4 we described the theory to reflect preferences with regards to concentration limits. When formalized this way, an existing profile can be checked to see if it satisfies the given concentration limits with a simple model expansion. If one of the limits is breached, no model will be found, and the *explain unsat* inference task can be used to find which limit is breached.

Theory comparison Active investors will typically update their profile regularly. In this case an automated comparison of two versions of the profile is helpful to ensure that correct amendments have been made. With the *model expansion* inference the logical equivalence of two theories can be checked by merging two theories T_1 and T_2, with concepts $Eligible1$ and $Eligible2$ respectively, such that $T_3 = T_1 \cup T_2$ and adding the constraint that an asset can only be eligible in one of both theories. If no model I_t that satisfies T_3 is found, the two theories are equivalent. If differences are found, the model expansion inference will return the model that contains these differences, which allows their manual validation.

Appendix A creation In the requirements we described the importance of the Appendix A for the company: this document is used on one hand as a contract with the client, and as input for the system that effectively assigns the collateral on the other hand. As our prototype was put in production in several runs, it was of major importance for the company that the manual creation of the Appendix A could be mimicked by our application.

The Appendix A consists of multiple parts: legal clauses, general information, eligibility criteria, haircut criteria, concentration limits, ... The table of eligibility criteria consists of 6 columns: Order (in which the rows should be evaluated), Field (refers to the reference data of the asset), Oper (applicable operand, e.g., =, >), Value and Outcome. The outcome can be

- "Accepted", meaning that the selected values are not non-eligible. Put differently, these values are open for eligibility (however, without being declared eligible). Accepted combinations can be reformulated to read as Non-eligible combinations. E.g., with three existing currencies EUR , USD and GBP if GBP is accepted, it means that EUR and USD are non-eligible. Or put differently: all currencies except GBP are non-eligible. Acceptable rules can only be formulated on one field.
- "Not eligible": the selected values are not eligible. An asset that thas this value in this field cannot be selected for collateral anymore, no matter what the values of the other fields of the asset are;
- "Eligible": this contains an eligibility rule. The securities that are the outcome of this rule, can be used as collateral.

The first column contains the order in which the rules should be evaluated. However, the order simply increases from 1 to n with n the number of rules. The Appendix A is in essence a concatenated first hit table (see Figure 10.5): a first hit table in which

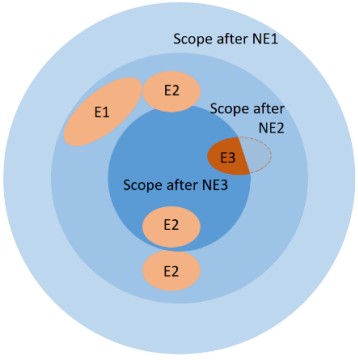
the *don't cares* are left out. The typical structure of the Appendix A is a number of

Eligibility				
F	Asset Type	Investment Grade	Country	Eligibility
1	Derivative	–	–	NotEligible
2	Fixed Income	True	BE	Eligible

Eligibility				
Order	Field	Oper	Value	Outcome
1	Asset Type	=	Derivative	NotEligible
2	Asset Type	=	Fixed Income	
	Investment Grade	=	True	
	Country	=	BE	Eligible

Figure 10.5: First hit table of investment profile and its Appendix A format

NotEligible rules, followed by some *Eligible* rules, then again some *NotEligible* rules and other *Eligible* rules. The *NotEligible* rules demarcate the scope of the *Eligible* rule. An example is shown in Figure 10.6, that visually shows the funnelling of the scope and the selection of eligible securities as in the adjacent table. Because of the consecutive execution of rules, all and only the securities in the orange parts are classified as *Eligible*.



Order	Rule	Outcome
1	NE1	Non Eligible
2	NE2	Non Eligible
3	E1	Eligible
4	E2	Eligible
5	NE3	Non Eligible
6	E3	Eligible

Figure 10.6: Visualisation of funnel effect of rule ordering.

To automatically create an Appendix A based on the collateral profile in FO(·), we created a four-step script.

- Create a dummy asset and assign the $NotEligible(asset) = true$. Use the *propagation* inference. The resulting propagations are the first row of the Appendix A and its outcome is *NotEligible*.
- Assign $Eligible(asset) = true$ to the dummy asset and use the *propagation* inference. The resulting propagations are the next row of the Appendix A and its outcome is *Eligible*.

- After these propagations, the *NotEligible* statements are evaluated rule by rule. A *NotEligible* rule is only added to the Appendix A, if it is not already entailed in the *NotEligible* rule detected in the first step.
- Finally, the *Eligible* statements are evaluated rule by rule. Only the rules that are not yet entailed in the *Eligible* rule detected in step two, are added to the Appendix A.

The result is an Appendix A that is automatically created within seconds after the client inputs his profile.

10.1.5 Evaluation

As the saying goes: the proof of the pudding is in the eating. The inference tasks were showcased to the company in a prototype. Following this, the company has launched a project to further develop this prototype into a production application. The first technical release in production was done in February 2021 and a second release with improved workflow for signing the profiles between counterparties was released in June 2021. As of the second release, clients from large investment banks have access to a sandbox environment for training purposes. The commercial roll-out was done in September 2021. The target users for this commercial release are operations teams in the treasury back offices of large investment banks globally (target around 500 users across 150 organisations). The correctness of the knowledge base was ensured by performing empirical tests. The testing of eligibility of assets against the profile has been done for descriptions with up to 20 rules, applied to portfolios of up to 300 assets with response times less than 3 seconds. These represent reasonable tranches for proper business use. Any larger portfolios can be tested off line with reporting being sent when processing has finished.

Profiles of 20 lines in NL might take up to 2000 to 3000 “if-then-else” statements when entered into the legacy systems. This is due to the large number of attributes (up to 150), the long lists of values behind most of these attributes (asset taxonomies, lists of countries, currencies, industries, ratings, issuers, etc.) and many overlaps and gaps in terms of eligibility. The “if-then-else” approach of the legacy system forces a sequential way of organising the information in the profiles, leading to thousands of statements. Our FO(·)-based approach allows for an almost line-by-line translation of the statements into logic. This leads to increased self-service by end clients, less operational errors and facilitated maintenance.

10.2 Case 2: Railway communication application

The second case was executed with Televic Rail, a division of Televic that focuses on passenger information systems and control systems for trains and lightrail vehicles. Televic produces the hardware, such as different kinds of internal and external screens, and provides the accompanying software that allows their clients to select the message to be shown.

Contributions In this case we demonstrate the suitability of FO(·) and DMN models by modelling the preferences of one specific client. We learn from this case that a standardised DMN model helps to detect inconsistencies. We demonstrate how the addition of background information leads to better completeness checking of decision tables in the interactive consultant. We evaluate three different ways in which the client can express his communication template (DMN, NL, PCS).

10.2.1 Requirements

In this case we look at the program that determines which messages are shown to passengers during a trip. The selection of the correct message is entirely the responsibility of Televic's client. Televic offers a GUI that clients can use to create templates for each message type. The template contains a set of rules that determine which message should be played. There is currently no way of controlling that the rules are complete, not overlapping and relevant. This leads to unexpected behaviour in the field, for example when a message announcing the arrival at the next station is played before the message when leaving the current station. Therefore, Televic wants to find a solution to allow clients to keep an overview of the rules they created and automatically detect conflicts in templates. An additional difficulty are unforeseen circumstances in the field. For example, the distance between two stations may be too short to play all selected messages. The train conductor might need to make live announcements that interfere with the foreseen messages. We want the system to react appropriately to this kind of situations, for example by cancelling or replacing some messages.

10.2.2 Maintaining overview of templates

Based on an example for one client, we created a DMN model of this client's template. It consists of six decision tables. One table is used to determine the sequence of the train, see Figure 10.7. The sequence is a concept that was introduced to reflect the position of the train relative to the station type and station location on its journey. It determines for a large part which messages will be played.

The five other tables each represent an output medium, and the messages that are selected for this medium based on the sequence. We could also opt for one large

Train Sequence					
U	Station Type	Station Location	Position	Trigger	Train Sequence
1	–	origin	at platform	driver id	1
2	major	–	after departure	axle rotation	2
3	not major	intermediate	after departure	axle rotation	3
4	–	–	between stations	termination of previous sequence	4
5	–	intermediate	before arrival	gps location	5a
6	–	terminating	before arrival	gps location	5b
7	airport	–	before arrival	gps location	5c
8	–	intermediate	at platform	door release	6

Figure 10.7: Decision table on train sequence

message output table, in which the output medium represents an additional column. However, as the variables to select the output message are not the same for the different media, we choose to create different tables with a limited number of variables: the largest decision table only contains seven variables.

Creating a DMN model of the client's request revealed several things. To start with, when putting the information in a standardised DMN format, it became clear that some combinations were missing, and other pieces were superfluous. For example, message A needs to be displayed if time to arrival in next station is more than 601 seconds, message B needs to be displayed if time to arrival in next station is less than 600 seconds. A definition for 600 and 601 seconds before arrival is missing. Depending on how the logic is implemented in the system, this may be a problem or not. An example of superfluous information is the content of an airport information movie, while the template aims to select the movie and does not focus on its contents. Moreover, the concepts used in the clients' description do not necessarily represent base concepts that can be used to create a template. The categorization of train stations is an evident example of this. A particular station can be described by some of its characteristics: its size, services, etc. This results in the distinction between major, minor and airport stations. A station can also be categorised in relation to its function in a particular journey: it can be the origin, intermediate or terminating station. In the clients' description, the station type combines both characteristics, so the type can be origin, minor, major, etc. In KBS the aim is to create a KB that describes the domain concisely and accurately. Hence, the terminology of the client cannot be adopted straight through. The knowledge engineer will need to analyse the domain independently, and map it with the client's concepts. Presumably, links can be defined between the two categorizations. For example, we assume that origin and terminating stations are never minor stations. The knowledge engineer will need to probe further for this, to create a complete KB. The resulting DMN model can be verified manually by the clients, or they can be translated automatically to a FO($\cdot$) specification. This in turn can be used in the interactive consultant to automatically verify the completeness of the decision tables, taking into account the possible values

of each variable.

In short the current process to create and validate new templates consist of the following steps:

- The client sends his message selection criteria to the company in an unstructured format.
- An operator at the company interprets the request, and translates it to DMN tables.
- The DMN tables are manually validated and verified by the client, or automatically translated to FO(·) and automatically verified.
- Based on the DMN tables, templates that steer the working of the system are created.

This way of working has several disadvantages. To start with, it requires a substantial effort from the operator to interpret the client's request and translate it to templates. Moreover, although DMN tables are useful for validation, verification and to keep overview, their creation is a time and resource consuming task. Therefore we propose to combine DMN models with NLP.

10.2.3 Creating new templates

To overcome the burden of operators that need to interpret the requests from clients, we use CNL. In line with the application described in Section 10.1 we use a tuple tree that represents the possible combinations of criteria that can be used to select a particular message. In the interface clients can create as many sentences as needed. However, the combinations specified in the tuple tree limit the variance. For example, if the station location is "origin", the position cannot be "before arrival". A typical sentence has the structure

$$\varphi_1 \implies \varphi_2. \tag{10.4}$$

with φ_1 a conjunction of literals that represent the selection criteria and φ_2 a message consisting of a conjunction of parts. The parts of a message need to be played in the correct order. This can either be implemented by creating background information on the generic structure of a message, or by adding an explicit order in the natural language sentence. As before, the sentences are automatically translated to FO(·) sentences. The resulting KB can be used in different ways.

Decision table creation One of the difficulties of the current way of working, is the lack of oversight of the templates that have been entered. This can be remediated by creating a DMN tables of the profiles. The *Show all models* inference task, returns the created sentences in the format of a unique hit table.

Decision model verification By using the completeness checking inference task, the completeness of the model will be checked. This is based on the domain of the variables, but also on the possible combination of variables that are determined in underlying tables. For example, if an origin station cannot be a minor station as reflected in the Background Information table, no template will need be created for this combination.

Background Information		
C	Station Location	Station Type
1	Origin	Major
2	Origin	Airport
3	Intermediate	Minor
4	Intermediate	Major
5	Intermediate	Airport
6	Terminus	Major
7	Terminus	Airport

Figure 10.8: Background information on station type and location.

Visualisation As the type of stations is limited, as well as the possible positions of the train, a visualisation can be added that shows the user which combinations were defined. In the example of Figure 10.9 there are three types of station (Major, Minor, Airport), three station locations (Origin, Intermediate, Terminating) and four positions (before arrival, at platform, after departure, in between). With some limitations (the train cannot be "before arrival" of the origin station, the "in between" messages are supposed to be the same regardless of the stations between which the train is located), this can be visualised in Figure 10.9. This shows that the templates for origin at platform, after departure from origin, between stations and before arrival at major station have been created.

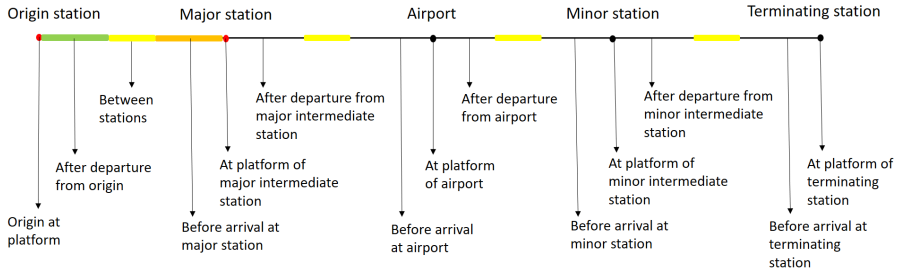


Figure 10.9: Visualisation of created templates.

10.2.4 Deciding between conflicting messages

The application's aim is to determine which message should be played at which moment during an actual train journey. This means that there might be practical restrictions to the general template. We have investigated two restrictions: 1) If stations follow each other closely, there might not be enough time to play the entire message. 2) In case of unforeseen events, the railway guard might create an announcement that should take precedence over other messages.

Limited time between two stations

Assume a situation in which three kinds of messages are played in the stage from one station to the other:

- The after departure message from the first station – assume this takes 100 seconds.
- The in between message – assume this takes 90 seconds.
- and the before arrival message for the second station – assume this takes 30 seconds.

These messages should not follow one another straight away, so at least 60 seconds pause between two messages should be respected. If the travelling time between the two stations is less than 340 seconds, a message selection strategy should be developed. Basically there are two possibilities: either some messages are cancelled, or the messages are shortened. Of course, a combination of both strategies is possible.

Cancel messages One of the possibilities is to cancel messages. Of course, the client should be able to decide which message should be cancelled. This can be done by assigning a priority to each of the three messages. The following specification can be used to determine which message should be played:

```
vocabulary V{
    type Station constructed from {station1, station2}
    type Time isa int
    type Messages isa int
    type Priority constructed from {low, medium, high}
    TravelTime(Station, Station):Time
    MessageDuration(Messages):Time
    ShortenedDuration:Time
    TotalDuration:Time
    priority(Messages): Priority
    NotPlay(Messages)
    Play(Messages)
}
```

```
theory T:V {
    ¬Play(m) ⟺ NotPlay(m).
    ∀ m:priority(m)=high ∧ MessageDuration(m)<TravelTime(s1,s2)
        ⇒ Play(m).
     ∀ m m2 s1 s2: priority(m)=high ∧ priority(m2) = medium ∧
        (MessageDuration(m)+MessageDuration(m2)) < TravelTime(s1,s2)
        ⇒ Play(m) ∧ Play(m2).
    ∀ s1 s2: (sum{m t:MessageDuration(m)=t:t}) < TravelTime(s1,s2)
        ⇒ ¬NotPlay(m).
    ∀ m1 m2: priority(m1)=high ∧ priority(m2)=medium ∧ NotPlay(m1)
        ⇒ NotPlay(m2).
    ∀ m1 m2: priority(m1)=medium ∧ priority(m2)=low ∧ NotPlay(m1)
        ⇒ NotPlay(m2).
    ∀ s1 s2: s1 ≠ s2
        ⇒ (sum{m t:MessageDuration(m)=t ∧ Play(m):t})
        ⩽ TravelTime(s1,s2).
}
structure S:V {
    Time = {0..350}
    Messages = {1..3}
    MessageDuration = {1, 100; 2, 90; 3, 30}
    TravelTime = {station1,station2,110; station2,station1,110;
        station1,station1,0; station2,station2,0}
    priority = {3,high; 1,medium; 2,low}
}
```

Shorten messages The other possibility is to compose the three messages from different building blocks, such that the entire message becomes smaller, and might fit in the total travel time between two stations. There are two ways to shorten messages: either some of the components that are part of the message (greeting, station list, next station,...) can be left out, or a shorter message can be used (e.g., only summing up the major stations instead of all stations). In the latter case, the algorithm needs to choose the correct messages from alternative ones. We assume that for each message type a shorter version exists. If only one message is defined for a specific sequence, it is used for both the normal and short version of the message. In the example below, this is the case for message 6, that is a copy of message 5. If no message is defined for a sequence, two message numbers with duration 0 are created.

```
vocabulary V{
    type Station constructed from {station1, station2}
    type Time isa int
    type Messages isa int
    type StationType constructed from {major, minor}
    type StationLocation constructed from {origin,intermediate,
        end}
    type Position constructed from {after_departure,in_between,
```

```
        before_arrival}
    stationType : StationType
    stationLocation : StationLocation
    position : Position
    TravelTime(Station, Station):Time
    Duration(Messages):Time
    TotalDurationNormal:Time
    TotalDurationMedium:Time
    TotalDurationShort:Time
    TotalDurationShortest:Time
    PriorityHigh(Messages)
    PriorityMedium(Messages)
    PriorityLow(Messages)
    Normal(Messages)
    Short(Messages)
    Selected(Messages)

}
theory T:V {
//****Sequence of messages****
   !m [Messages]: m = 1 ∨ m = 2 ⇐ stationType = minor ∧
         stationLocation = intermediate ∧ position = after_departure.
   !m [Messages]: m = 3 ∨ m = 4 ⇐ stationType = minor ∧
        stationLocation = intermediate ∧ position = in_between.
   !m [Messages]: m = 5 ∨ m = 6 ⇐ stationType = minor ∧
        stationLocation = intermediate ∧ position = before_arrival.

//****of each type, one message needs to be selected****

¬(Selected(1) ∧ Selected(2)).
¬(Selected(3) ∧ Selected(4)).
¬(Selected(5) ∧ Selected(6)).
(Selected(1) ∨ Selected(2)).
(Selected(3) ∨ Selected(4)).
(Selected(5) ∨ Selected(6))..

TotalDurationNormal = sum{m : Normal(m):Duration(m)}.
TotalDurationMedium = sum{m : PriorityLow(m) ∧ Short(m):Duration(m)}
        +sum{m :PriorityMedium(m) ∧ Normal(m):Duration(m)}
        + sum{m :PriorityHigh(m) ∧ Normal(m):Duration(m)}.
TotalDurationShort = sum{m : PriorityLow(m) ∧ Short(m):Duration(m)}
        +sum{m :PriorityMedium(m) ∧ Short(m):Duration(m)}
        + sum{m :PriorityHigh(m) ∧ Normal(m):Duration(m)}.
TotalDurationShortest = sum{m : Short(m):Duration(m)}.
```

```
    !m: TravelTime(station1,station2) > TotalDurationNormal ∧
        Normal(m) ⇒ Selected(m).
    !m : TotalDurationMedium < TravelTime(station1,station2) ⩽
        TotalDurationNormal ∧ PriorityLow(m) ∧ Short(m)
        ⇒Selected(m).
    !m : TotalDurationMedium < TravelTime(station1,station2) ⩽
        TotalDurationNormal ∧ ~PriorityLow(m) ∧ Normal(m)
        ⇒Selected(m).
    !m : TotalDurationShort < TravelTime(station1,station2) ⩽
        TotalDurationMedium ∧ PriorityLow(m) ∧ Short(m)
        ⇒Selected(m).
    !m : TotalDurationShort < TravelTime(station1,station2) ⩽
        TotalDurationMedium ∧ PriorityMedium(m) ∧ Short(m)
        ⇒Selected(m).
    !m : TotalDurationShort < TravelTime(station1,station2) ⩽
        TotalDurationMedium ∧ PriorityHigh(m) ∧ Normal(m)
        ⇒Selected(m).
    !m : TravelTime(station1,station2) ⩽ TotalDurationShort ∧
        Short(m) ⇒ Selected(m).
}
structure S:V {
    Time = {0..350}
    Messages = {1..6}
    MessageDuration = {1, 100; 2, 90; 3, 90; 4, 60; 5,30; 6,30}
    TravelTime = {station1, station2,209; station2, station1, 300;
        station1, station1, 0; station2, station2, 0}
    PriorityHigh = {3;4}
    PriorityMedium = {1;2}
    PriorityLow = {5}
    Normal = {1; 3; 5}
    Short = {2;4;6}
}
```

On-board announcements A train journey might not run as planned, and the railway guard might want to make an announcement. This announcement should take priority over the planned messages. The rules could be as follows:

- If the journey of the train has changed, all remaining announcements should be cancelled. Instead an apology message should appear on the screens, telling passengers to follow the on-board communication of the railway guard.
- If the journey stays the same, the planned messages are still accurate, and should be displayed as planned if there is enough time.
- However, due to the additional announcement the available time might be too short. In that case the system should decide which message to shorten/cancel in line with the possibilities described above.

10.2.5 Using a product configuration system

The current way of working is unstructured: the way in which templates can be communicated to the company is not standardised, and every available parameter can be used. Luckily, the list of possible parameters is relatively limited to 32 parameters. Moreover, the communication with passengers is currently mainly focused on travelling information and similar across transportation means and companies. Instead of leaving complete freedom to clients, another possibility is to create the communication templates in a configurator. In Chapter 7 we defined a product configuration system as "a system to support the design of products that consist of multiple components in multiple variants by formulating constraints on how these parts can be combined to arrive at a product with the required characteristics." A single template consists of multiple message components (greeting, calling pattern,...) in different variants (e.g., morning greeting, evening greeting). There exist constraints on how these components can be combined. For example, only a single greeting can be selected, but in the announcement category more messages can be selected. The constraints limit the number of possibilities, and hence the complexity of the task.

We created a first prototype of a train communication PCS in the Interactive Consultant (Figure 10.10). It divides the relevant parameters in multiple blocks. The first block contains the message reference and the medium of the message. The message reference is the unique identifier of the message. The medium defines if and which screen (internal, external), or audio the template is for, if it is a short message or not, and the priority of the message. The second block determines the sequence for which the template is used. The third block contains optional selection criteria, such as the distance to the next station or the time of the day. These three first blocks aim to define the conditions in which to play the message that is selected in the fourth block, the message components block. In the components block, clients can choose at most one message from each message component (in the shown example, the components are "greeting", "station list" and "next station"). One of the internal constraints of the configurator, is that the selected messages are played in a predefined order (i.e., greeting before station list and next station, station list before next station). When the user has selected the values and messages for one template, it should be added to the KB. In this specific case this is possible because rules have the form of an implication, with conjunctions on both the right and the left side (see 10.4). Moreover, we know that the right side represents the choices from the fourth block, all the other choices are part of the left-side conjunction. Hence, the rule to add to the KB can be constructed as follows: i) the atoms with truth value *true* from blocks one through three are part of a conjunction that represents the if-clause of an implication. ii) the atoms with truth value *true* from the fourth block are part of a conjunction that represents the then-clause of an implication.

In the example of Figure 10.10, the template reads:

$$\begin{aligned}\text{Message} =1 \wedge \text{Medium} = \text{InternalScreen} \wedge \\ \text{stationType}(1)= \text{major} \wedge \text{position}(1)= \text{beforearrival} \wedge \\ \text{stationLocation}(1)=\text{intermediate} \wedge \text{timeOfDay}(1)=\text{beforeNoon}\end{aligned}$$

```
⇒ greeting(1,InternalScreen,morning)∧
stationList(1,InternalScreen,majorstation).
```

As the screen becomes cluttered if all the possible combinations of predicates are shown in each block, it would be a major improvement to show only the predicate combinations with the selected values from the first block. Figure 10.11 shows a mock-up of the PCS. When the user clicks the "Add Rule" button, an overlay window pops up that shows the sentence that is added to the KB in FO(·). After closing the pop up, the user returns to the initial screen with all values set to unknown.

10.2.6 Conclusion

The creation of communication templates typically fits in a larger project of acquiring and installing a new communication system. The exact content of the messages that are shown is a concern that needs to be addressed occasionally on a project basis. There is a chance that this area is relatively new for the client. Televic on the other hand, is specialised in developing these communication systems, and has installed them at numerous clients. Although they cannot know the content of the communication templates, the company is very familiar with the possible parameters that can be used in these templates. They have a good overview of the domain. Televic's communication system is very flexible to configure. This also means that for inexperienced clients that have no exotic requests, the modelling effort might be perceived as burdensome. In this case we proposed a number of tools to alleviate the burden. The use of DMN supports the verification and validation of the set of templates. CNL provides a user-friendly and intuitive way to create sentences to describe a template. These sentences can be translated automatically to FO(·), that can be used to create comprehensible visualisation, e.g., by using the inference of decision model creation. Other inferences, such as decision model completeness checking support the validation and verification of the templates. As Televic has the expertise in railway communication, the development of a PCS for template creation is another possibility. This reduces the complexity for clients, which improves the template creation process, although this limits the flexibility for clients. Combining techniques, such as the use of CNL for template creation and the use of an IDP inference task to make a DMN table that can easily be verified by the client, facilitates template creation. Because the format of the rules standardised to 10.4 with the purpose to select messages, it is also possible to create a template configurator in the Interactive Consultant. The effective implementation of such a PCS is part of future work.

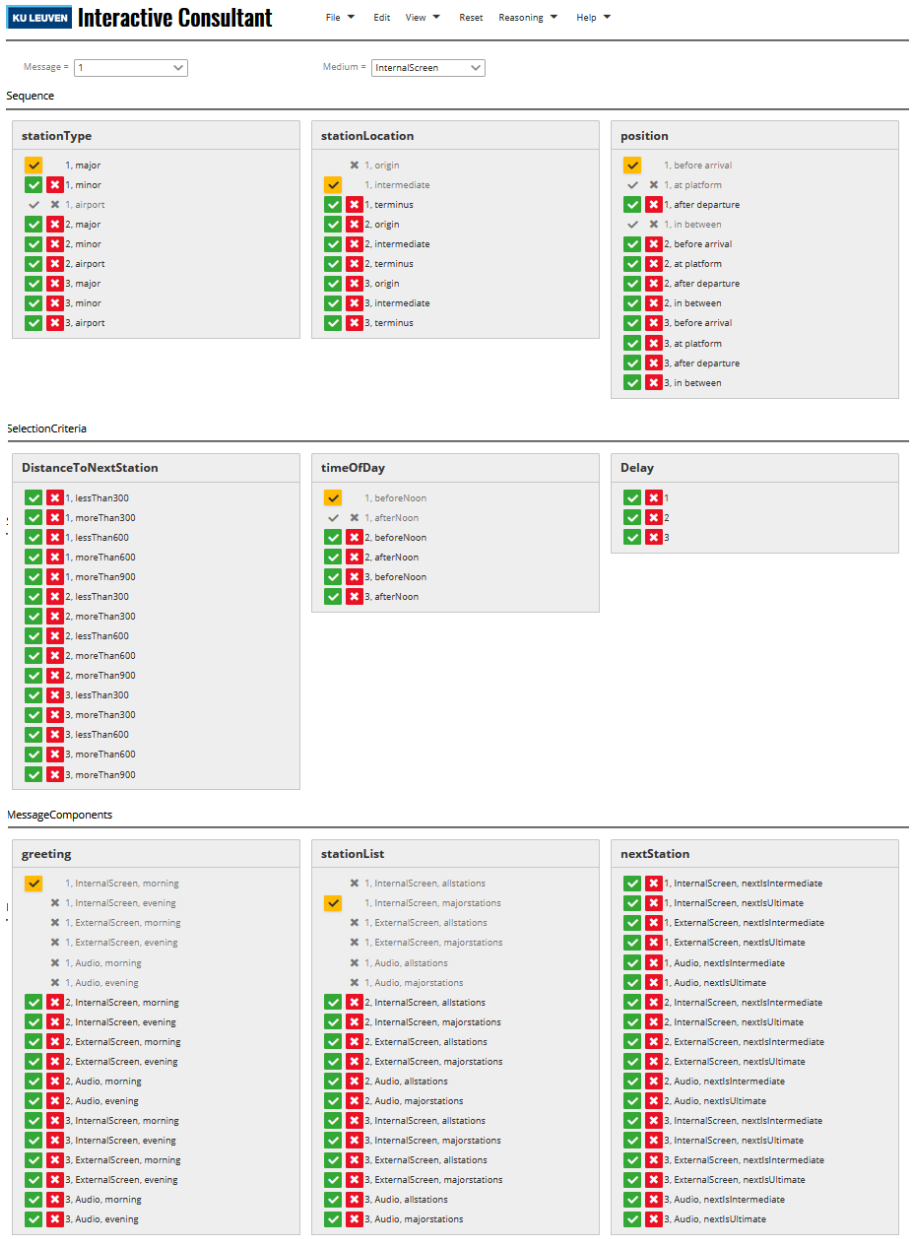


Figure 10.10: Truncated screenshot of message selection PCS.

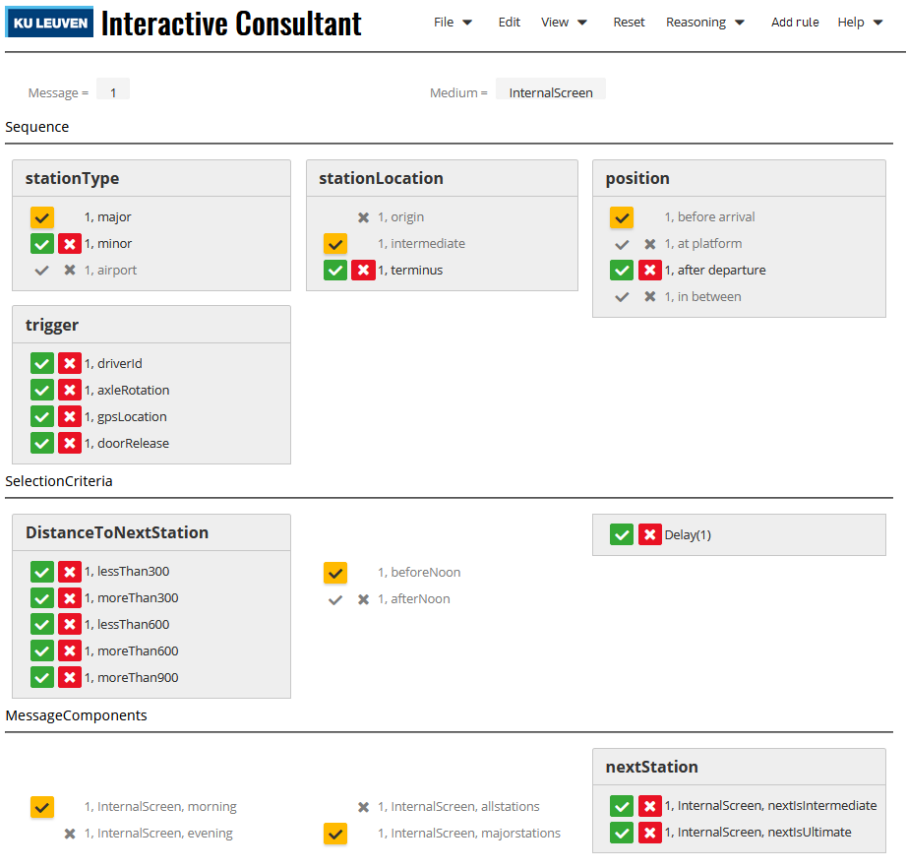


Figure 10.11: Prototype of message creation PCS.

10.3 Case 3: Employee's time registration

This third case was provided by Teal Partners. It concerns the wage calculation of employees, based on their time registrations.

Contributions In this case we modelled the time registration problem in two ways in FO(·). The first way is the traditional approach using intervals in which a clocking-in is associated with a clocking-out. We also developed an event calculus model, in which the status of a specific point in time is described by the actions that take place. As both models are possible, we pinpoint in this case the differences between the two. We demonstrate how a generic system can be developed and tailored to the needs of individual clients by making the KB accessible through cDMN tables.

10.3.1 Introduction

The use of automated time registration for employees based on their clocking is common practice. The data gathered this way, needs subsequent processing. Some working time performed by the employee will not be reimbursed, while other working time will receive a bonus or augmented payment. The rules that govern the payment may come from three sources: the government, sectoral agreements, and company specific labour contracts. The purpose of the case study is threefold. The main focus is on trying if the regulations can be formulated in FO(·). Second, we want to test if this can lead to a workable system to calculate wages based on time registrations. Finally we investigate if we can find a representation of the KB such that it can easily be adapted. Time registration calculation is a general problem, but the exact rules are different for every company. Hence, the system would need to be flexible and understandable enough to be adapted by consultants for every company.

An application would need to be able to address several complexities, some of which are described below.

- Different time definitions: some of the rules are formulated as clock on the wall timings (e.g., worked hours before 8 A.M. are not taken into account), while others use time durations (e.g., after 8 worked hours, there is an overtime payment). Time periods under consideration vary from per day, per week, per month, per quarter or even per year basis.
- Multiple interpretations per interval, e.g., one part is paid 100%, while another part is paid 150%.
- Synthetically created intervals: when the legally obliged lunch break is not taken, a virtual lunch break (during which employees are not paid), needs to be created in between the existing interval.

To find out if the regulations can be modelled in FO(·), we modelled the following rules for a schedule that spans two days (from 16:00 on day 1 to 0:30 on day 2):

1. If first clocking of the day is between 0:00 and 16:00, the hours in this zone do not count.
2. If last clocking of the day is between 0:30 and 0:45, the hours in this zone do not count.
3. A break can be taken at the earliest 1 hour after the first clocking-in.
4. An employee may not work more than 6 hours without having taken 30-minute break. If no break was taken, it is forced, ending at the end of the sixth hour.
5. A break may be taken in two parts, of each at least 15 minutes.
6. Intervals of less than 15 minutes "not worked" are not part of a break.
7. Mandatory additional 15-minute break if user works more than 9 hours. If no break was taken, it is forced, ending at the end of the ninth hour.
8. Hours worked between 22:00 and 6:00 receive a night premium.
9. Hours worked between 20:00 and 22:00 receive an evening premium. Consecutively worked hours that go from evening to night will become night-hours if the amount of night-hours is larger than the consecutive evening hours
10. If worked more than 8 hours but less than 9 hours, 100% overtime is assigned.
11. If worked more than 9 hours, an overtime of 150% is assigned.

10.3.2 Modelling the rules

The first objective of this case study is to discover if the FO(·) language is expressive enough to model the time registration rules. We compared two approaches: interval modelling and fluents and action modelling.

Interval modelling

In the interval modelling approach we reason based on intervals. Each clocking-in is linked with its associated clocking-out to create an interval, that can be assigned a unique identifier. This kind of modelling is convenient for use in straightforward situation, such as the one reflected in the upper panel of Figure 10.12. However, if virtual intervals need to be added, as in the lower panel of Figure 10.12, for example if the employee did not take a lunch break, if they worked overtime or changed cost centres without clocking, this is difficult to do in the interval based approach. Although we successfully created an interval-based model, it quickly becomes difficult to read. For example, the worked hours are the sum of the intervals:

```
Presence = sum{i:TypeInterval(i) = Work : Duration(i)}.
```

However, some parts of the interval should not be counted:

```
!i : Duration(i) = (ClockOut(i)−ClockIn(i)) − Disqualified(i).
```

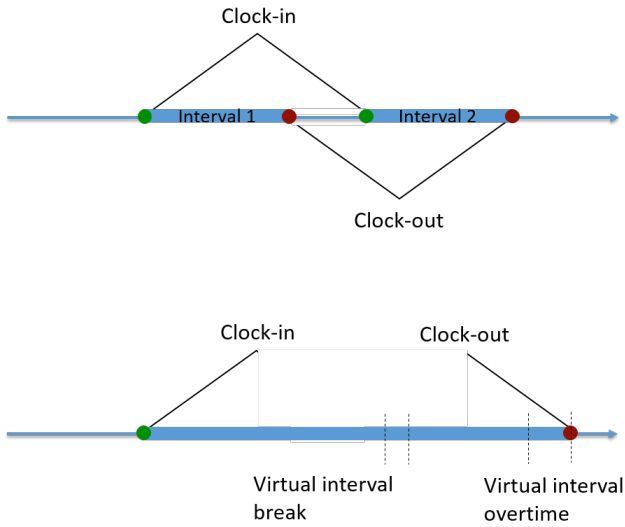


Figure 10.12: Interval-based modelling.

The reason for the disqualification can be the clocking-in before 16:00 or between 00:30 and 0:45. The model is built with minutes as time unit (so 4PM equals 960 minutes).

```
{disqualified(1) ← ClockIn(1) < 960.
∀ i: disqualified(i) ← 1470 < ClockOut(i) < 1485 .  }

ClockIn(1) < 960 ∧ disqualified(1) ⇔
        Disqualified(1) = 960 − ClockIn(1).
∀ i: 1470 < ClockOut(i) < 1485 ∧ disqualified(i) ⇔
        Disqualified(i) = ClockOut(i) − 1470.
∀ i: ¬ disqualified(i) ⇒ Disqualified(i) = 0.
```

Event calculus: fluents and action modelling

Another way to model time registration is by using event calculus for logic programs as proposed by Kowalski and Sergot in their 1986 paper [82]. In this approach the fluent represents a certain status, and actions can change this status. The initial status of an employee is "not at work". If the action of "clocking in" happens, this might change to "at work". However, if the employee clocks in before 16h00, the status changes only to "at work" at 16h00. In Figure 10.13 the actions, that happen at a specific point in time, are shown as dots. The fluents are the status over a period of time, shown with the curly brackets under the arrow. The figure below shows the

situation where the employee did not take any lunch break. As rule 4 commands that 30 minute lunch break needs to be taken, an automatic lunch break is assigned the last half hour before working six hours. An automatic lunch start action is generated after 5,5 hours at work, and an automatic lunch end is generated after 6 hours at work. As per definition, the period in between is registered as break.

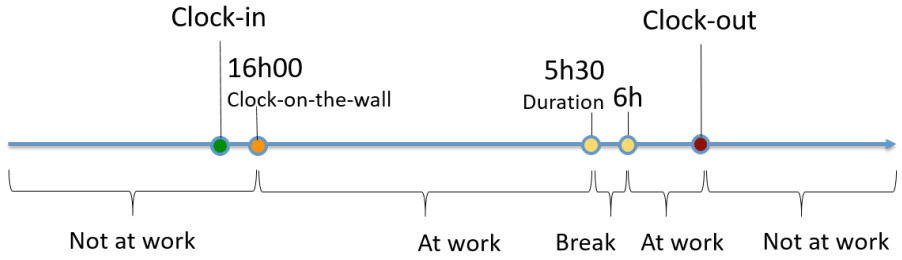


Figure 10.13: Fluents and action based modelling.

In the time registration case, we have two kinds of fluents: inertial fluents, that stay the same unless an action changes them, and time-dependent fluents, that depend on clock-on-the-wall time. There are also two kind of actions: external actions, that are initiated by the employee (e.g., clock-in, or change cost center), and automatic actions, that happen automatically by the system (e.g., start auto break). One part of the KB defines how fluents are defined based on the earlier fluent and actions. For example, at a specific point in time a certain fluent holds, if there occurs an action at that moment that initiates this fluent. A fluent can also hold at a specific point in time, if it held on the previous point in time, and there did not occur an action that ends the fluent. In FO(·), this is modelled as such:

```
{holds(fluent, time) ← ∃ action: occurs(action,time)
        ∧ initiates(action,fluent).
holds(fluent, time)  ← holds(fluent,time−1)
        ∧ ¬(∃ action: occurs(action,time)
        ∧ terminates(action,fluent)).}
```

Another part of the KB then models the exact rules applicable in a company. For example, employees are paid a night wage, if they are at work, and it is night:

```
{∀ time: paidAs(time,NightWage) ← holds(AtWork, time)
        ∧ holds(Night,time).}
```

Characteristics of event based calculus The example above demonstrates how a new fluent can be defined by simply adding one or more action(s). Therefore event calculus is said to use additive updates instead of deletions. If a relation no longer

holds, an action that marks the end of the relationship is added instead of deleting the entire relationship [82]. In that, past and future are treated symmetrically and actions can be added in any order [82]. In the example above, the clocking in and clocking out actions already happened before the automatic lunch start and end actions were added. The event calculus approach allows for multiple events to take place at the same time. E.g., if an employee works nine hours on a certain day, they are at work between clocking in and clocking out. As from the eight hour they are also in overtime. Moreover, as the status of a point in time is derived from the previous fluent and start action, this gives semantic meaning to the fluent, which can at the same time function as a check [82]. This kind of control is absent in interval calculations.

Tailoring the system The time registration calculation is unique for every company. Legislation forces certain rules, but also sectoral agreements impact labour regulation. Finally, company specific contracts result in a unique situation. The provider of time registration systems will need to tailor the generic software to the requirements of the client. Consequently, it is important that the KB can be adapted by consultants of the company, however without touching the logic of event calculus. Luckily, the KB can be split in a domain-independent part, that contains the definitions of fluents, and a domain specific part that contains the actual rules. The latter part is simple enough to be formalised in a cDMN model. The domain-independent part is developed by the servicing company, while the knowledge of the domain-dependent part is available at the third party only, and should be modelled by consultants familiar with the application.

Figure 10.14 depicts a high-level DRD of the studied case. The lower part is the generic part, and contains the logic to determine the status, based on the list of actions and fluents. The upper part is the company specific part that contains the rules that are applicable.

10.3.3 Evaluation

Modelling the time registration problem proved to be feasible in both the interval approach and the event calculus approach. However, the event calculus approach was found to be more straightforward to model. Some of the calculations are not related to clocking actions, such as the overtime calculations or night premiums. Moreover, in case of missing clockings, such as a missing lunch break, the event calculus only requires the calculation of the start and end time of the virtual break, whereas this requires the creation of multiple intervals in the interval approach. The developed prototype used ten minute time periods to perform calculations. In a real-life system the time period is much smaller, probably up to the second. Hence, system performance is an important point of attention to develop the prototype further into a production ready system.

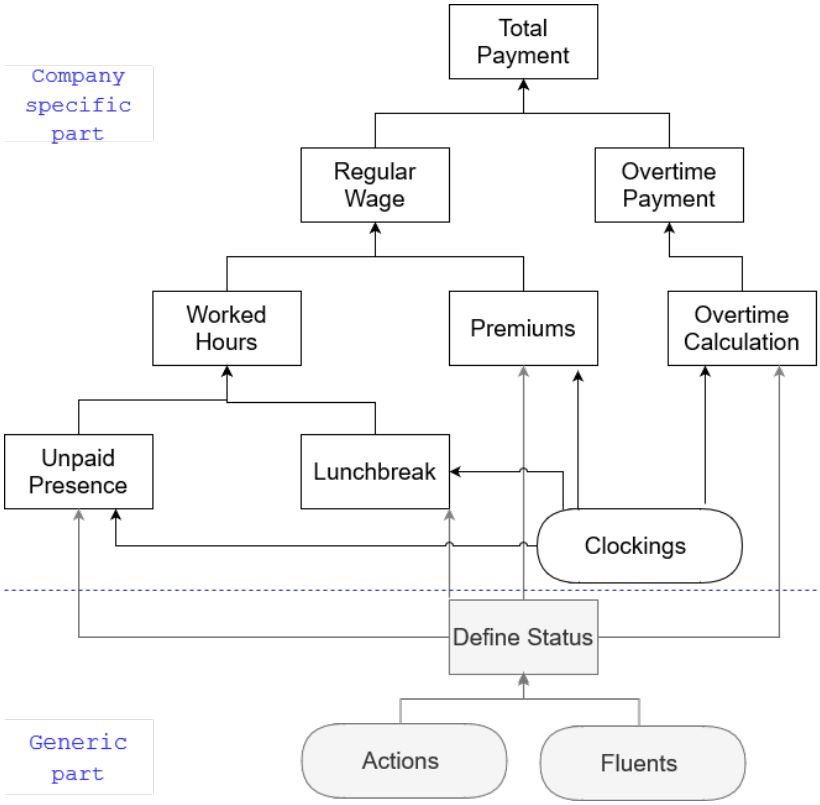


Figure 10.14: High-level DRD of time registration problem.

10.4 Generalizing findings

The cases described in the previous sections are third party applications because they support third parties in their decision making process, where at least a part of the knowledge is only available at and specific for the third party. Note that the specificity of knowledge at the client is crucial. If e.g. a legal application is developed commissioned by a third party, but the inherent knowledge is general, rather the guidelines for legal application should be followed.

In the three cases, a part of the knowledge that is considered crucial for the success of the application, is only available at the third party. This limits the in-depth analysis of the knowledge involved in each case. Nevertheless, the level of expertise of the third party relative to the expertise of the developing party is different for each of the three cases.

In the case of the collateral eligibility profile, the company has excellent expertise in the financial market inner workings, as do their clients. The knowledge to which the company has no access, is the actual profiles that their clients agree upon in their collateral agreements. Moreover, the company has a lot of clients who create regularly collateral profiles to collateralize new transactions with counterparties. In summary, this case is characterized by excellent domain knowledge by the company and third party, individual preferences that materialize in different collateral profiles and numerous clients with numerous profiles.

In the case of the passenger communication system, the expertise of the company versus the client is slightly different. Although both parties are acquainted with railway systems, they both have their own point of view. The company knows which parameters are available to create templates, and they have an overview of the different requirements for different clients. On the other hand, the client has expertise on the train journeys, and how they want to address passengers. They know the timings between stops, have their own communication style and obligations (e.g., contractual display of advertisements). The number of clients of the company is rather limited, as their clients consist of public transport providers. The clients have a relatively stable number of templates to create – once the communication system is installed, this is unlikely to change a lot. So the creation of templates likely happens on a project basis. This case is characterized by divided domain knowledge by the company and the third party, similar profiles that are created for different journeys, and a project-wise creation of communication templates.

The third case of time registration calculation contains a lot of complex information: time that can both be used as clock-on-the-wall time or as duration, constraints on daily, weekly or monthly basis, creation of virtual clockings,... The knowledge sources in this case are threefold: legislation, sectoral agreements and company contracts. The company that develops the application has access to the legislation and sectoral agreements, but as a standard not to company contracts. This means that the company needs to be involved for that part. For this case, there is no finite set of variables that is used to determine time registration calculation. This means that it is impossible to

build a complete tuple tree for the domain. Moreover, the fluents and action approach requires a specific set of rules that is difficult to formalise if one is not acquainted with the approach.

10.5 Recommendations

We have demonstrated in two case studies that the use of NLP and CNL can be very useful as support to create a KB. However, this approach should not be blindly adopted. The combination of NLP with an underlying reasoning engine such as IDP is recommended in case the developing company has a good overview of the domain and market practices, but with a large freedom for customers to create their own rules and constraints. The possible set of variables to use for it, and their domains, should be finite. This is necessary to be able to create a tuple tree that shows the appropriate building blocks to users. The resulting CNL sentences can be automatically translated to an FO(·) KB. The use of IDP as an implementation of the KBP allows the usage of various, generic inference tasks that support the validation and verification of the KB itself. Examples of this are the visualisation of (parts of) the KB, for example in decision tables; or the comparison of two profiles. Although the use of the NLP module that allows the user to type sentences in free format English has proven its merits, this does not mean that the use of building blocks to create CNL sentences should be underestimated. First, the CNL module is necessary to translate formatted sentences in formatted FO(·). Without it, the natural language would be too ambiguous to translate unambiguously to FO(·). Second, for users that are not used to expressing their constraints, the building blocks that are shown, function as a prompt to think of constraints they want to input.

We see less use for the combination of NLP and IDP where the number of variables and their possible values are not limited. In that case it becomes impossible to build a tuple tree that can cover all necessary information. Also in case where the third party knows only a part of the needed information, this approach seems less appropriate. In that case, a representation in cDMN with tables that have different read and write access, depending on the level of familiarisation with the system, is preferable. If the company has more expertise in the field than the customer, it might be a valuable option to investigate if the creation of a product configuration system would be a viable alternative to limit the client's freedom of choice.

10.6 Conclusion

Despite their diversity, third party applications have one important characteristic in common. I.e., a part of the necessary knowledge is only available at and specific for a third party. If the kind of knowledge that only the third party possesses is their preferences, this means that the developing company has or may obtain the relevant knowledge to create the ontology of the domain. The system they develop aims to

elicit preferences of the client, and should integrate the appropriate tools to this end. We found that the combination of a NL interface with an FO(·) KB offers many possibilities for preference elicitation, representation and validation. A logic engine can be used to develop many supporting services on top of the original task. If the third party also owns domain knowledge, such as independent company specific rules, it might not be possible for the developing company to have a complete view on the ontology of the domain. In this case the application should be tailored to the needs of the client, including the development of an extended ontology. The use of graphic models, such as a DRD, can be helpful to visualise which parts of the system can be adapted, and which parts are fixed.

Chapter 11

Knowledge complexity

In the previous chapters we discussed the kinds of knowledge of each application type. We did this using the distinction between domain knowledge, inference knowledge and task knowledge. Besides these kinds of knowledge, the complexity of the knowledge might be a distinguishing factor between different application types. Our goal is to get some insight on the complexity of a domain. Therefore this chapter focusses on the complexity of knowledge and how to measure it.

Contributions In the absence of an accepted, objective measure for the complexity of knowledge in a KB, we developed a KB complexity measure using the domain's ontology and parse trees for the sentences in the theory. The measurement automation has been developed by Simon Vandevelde. We compared the complexity of the main prototypes that we developed and integrated the results with the findings on application knowledge from the previous chapters. We indicate paths for further research in this area.

What is "complexity"? A definition can be found for the term "complexity", for example the complexity of a system has been described as the volume of interdependencies and degree of interaction between the elements of a system [93]. Any such definition fails to catch the multifaceted nature of the concept that lies at the heart of complexity research. As for the concept of "fairness", the concept of "complexity" depends on the viewpoint from which it is seen [94]. Complexity research knows many subfields, for example research on complexity of process and decision modelling formalisms (see [63] for some examples). The predominant conviction is that the complexity of a model directly impacts its readability and comprehensibility [63]. In this dissertation we want to compare the complexity of KB. This may have many elements, for example, the complexity can be considered different by FO($\cdot$) experts versus novices, the complexity to read a KB can be different from the complexity to

adjust or create it, the complexity of the KB may be different from the complexity of expert knowledge in this domain. We seek to use an objective measure of KB complexity independent of the user of the KB, and independent of the way the KB is used by inference tasks to calculate solutions.

11.1 Knowledge complexity in expert systems

Meyer and Culey [100] developed a framework to measure the complexity of an expert system. The measure is based on a combination of the complexity of the underlying knowledge at one hand and the technological complexity of the system on the other hand. As we focus on the complexity of knowledge, we limit ourselves to the first section of Meyer's equation. It consists of seven weighted criteria for which a score from one to three, four or five is assigned based on mostly qualitative criteria.

A first important criterion is the breadth of the domain. This refers to the distinct fields of expertise that are employed by the experts. The paper suggests that this equals the number of knowledge bases that exist within the given system. A score of one (for a single domain) to three (for more than two domains) is assigned for this criterion. The second criterion is the depth of these domains. The more education and experience one needs to have to be considered an expert in the area, the deeper the domain. A score from one to three is assigned. One means common knowledge, requiring no specific education and less than ten years of experience. Two means deep knowledge, requiring a specific degree (M.D. or PhD) or at least ten years of experience. Three is the deepest level of knowledge, requiring a specific degree *and* ten years of experience. Another important factor is the pace of change in a domain. A domain that is under continuous evolution is considered more complex than a stable domain. It is measured by the time that experts are still considered experts without updating their knowledge. If this is still the case after five years, the rate of change is low and a score of one is attributed. An expert of two years ago that did not add any knowledge and is still considered an expert today, operates in a moderate pace of change domain (score two). If remaining an expert requires continuous knowledge updating, it is a high change domain, and gets awarded a score of three. The fourth factor is the penetration of the domain. This is the portion of the domain that is covered by the system. If only a little portion of the expert's knowledge is used in the system, a score of one is awarded. If the complete expert's knowledge is incorporated, a score of five is assigned. The breadth of the information inputs constitutes the fifth factor. This criterion counts the number of inputs. The inputs are not the number of variables of the system, but refer to an entity of information input. An example are coal characteristics data, plant configuration information, operation cost and performance data in a coal advisor application [100]. The score ranges from one (1-2 inputs) to three (5 or more inputs). The next criterion measures the complexity of the inputs: they can either be used without additional interpretation (score 1), some (score 2), or a high (score 3) degree of interpretation needs to be used. At the output side of the system, the type of outputs are categorised in four domains: problem diagnostics, recommended actions, actual solutions and hypothesis testing.

Depending on the number of categories that is present in the system, a corresponding score from one to four is assigned. The table below summarises the criteria, their scores and weights.

	Breadth	Depth	Change	Penetration	Inputs	Ambiguity	Outputs
Weight	20	20	20	12	20	20	15
Score							
1	1 domain	Common	Low	Little	1 or 2	Low	1
2	2 domains	Deep	Moderate	Partial	3 or 4	Moderate	2
3	≥ 3 domains	Deepest	High	Moderate	$\geq$ 5	High	3
4	-	-	-	Substantial	-	-	4
5	-	-	-	Complete	-	-	-

Table 11.1: Overview of criteria and weights of knowledge complexity according to Meyer and Culey

Meyer and Culey divide the output by 4.2 to normalize the score to 100. They used their measure to calculate the complexity of 50 systems. The mean knowledge complexity of these cases is 57. Scores above 57 are considered "high", the ones below 57 are considered "low".

11.1.1 Knowledge complexity of case study applications

We have tested the approach of Meyer and Culey on the KBs that we developed for our case study applications.

Complexity of registration duty application We have summarised the scores according to the Meyer and Culey measure in Table 11.2.

Breath of the domain	1	Calculation of registration duties
Depth of the domain	2	Legal expert needs to hold a degree in law
Change of the domain	3	Legislation changes continuously
Penetration	3	Portion of legal knowledge in a legal application is complete, but this is only a part of the expert's knowledge.
Inputs	2	Information on the buyer(s) Information on the purchase Information on earlier purchases Information on the purpose of the purchase
Ambiguity	2	Interpretation of information from clients needed
Outputs	4	System can be used flexibly
Knowledge complexity	296	

Table 11.2: Knowledge complexity of our registration duty application.

Complexity of knowledge in an ETO PCS As mentioned before, traditional PCSs contain more complex knowledge than traditional software systems [56]. We already established in the introduction that the additional amount of domain knowledge necessary for the creation of KBSs, leads to the inclusion of more complex knowledge in a KBS than in a traditional IT system, even if the knowledge is harder to purify from the latter. In this section we investigate if KBs of PCSs are more complex than other KBSs. Intuitively this can be expected. After all, for each additional component, the number of possible variants and rules and constraints to be included, increases exponentially [7]. The purpose of PCSs is to offer *more* complexity (i.e., product variants), but keep this complexity better manageable. Such exponential complexity growth is probably not present in many other application types.

An overview of the complexity criteria as proposed by Meyer and Culey and their score for an ETO PCS is shown in Table 11.3. Some of the criteria did not receive a single score, but a range. The breadth of the domain, for example, counts the number of KBs. However, the way the knowledge is formalised in one or more KBs is to some extent a subjective judgement of the modeller, instead of specific to the domain. A modeller who splits the same information over different KBs will render the information more readable, but it would increase knowledge complexity according to this measure. Information on shrinkage calculation or material strength could be split in a different KB, but can also be incorporated in the central product design KB. Also the description of what an "input unit" is, is not univocal, hence the scoring range. The only variable input unit are the clients request. However, the other factors need to be taken into account when proposing a design. This kind of information is cited by Meyer and Culey as information units in their example, hence the inclusion in our score.

Breadth of the domain	1-2	Design of product Knowledge of materials
Depth of the domain	2	Product engineers with engineering degree
Change of the domain	3	Market and technology changes continuously
Penetration	4	Expert needs to combine many fields of expertise
Inputs	2-3	Clients requests Historical designs Installation conditions Production costs Market practices
Ambiguity	2	Evaluate suitability of historical design
Outputs	4	System can be used flexibly
Knowledge complexity	308-348	

Table 11.3: Knowledge complexity of an engineer-to-order PCS.

Knowledge complexity of scheduling application The complexity score of scheduling applications is reflected in Table 11.4. The main complexity from a knowledge point of view is the elicitation of preferences from the employees and

Breadth of the domain	1	Creating a planning
Depth of the domain	1	No degree or experience needed
Change of the domain	1	Preferences change, but not procedure to create schedule
Penetration	1	Side task of head doctor
Inputs	1	Hospital policy Employee preferences
Ambiguity	3	Difficulty to elicit preferences from employees
Outputs	4	System can be used flexibly
Knowledge complexity	212	

Table 11.4: Knowledge complexity of a scheduling applications.

interpret them. For the scheduling itself the complexity is to find a schedule that reconciles the interests of different stakeholders, but this is done automatically by the inference task.

Knowledge complexity of third party applications In Tables 11.5 to 11.7 the knowledge complexity score is calculated for each of the three cases. The expectations are that the scores for this kind of applications will be lower, because they do not contain the knowledge that is present at the third party.

For the application that determines the eligibility of assets for collateral selection, we take the viewpoint of the operator that previously created the collateral profiles manually. Such an operator typically has a professional degree, and becomes an expert through on-the-job learning.

Breadth of the domain	1	Collateral selection
Depth of the domain	1	Internal training of operators
Change of the domain	1	Available characteristics of assets is fixed
Penetration	3	Limited other tasks for operators
Inputs	1	Client requests
Ambiguity	2	Evaluate client requests
Outputs	4	System can be used flexibly
Knowledge complexity	216	

Table 11.5: Knowledge complexity of collateral selection application.

The hard and software of the train communication system is offered to multiple clients by the developing company. Clients can pass across their preferences in any format. An operator of the developing country translates this into the templates that are used to play the messages.

The time registration case is a third party application in which the entire system is tailored to the needs of the client. The knowledge sources are legislation, sectoral agreements and company contracts. An expert from the client will need to collaborate with an expert from the developing company to obtain a satisfactory result.

Breadth of the domain	1	Creation communication templates
Depth of the domain	1	No degree or long experience required
Change of the domain	2	Change in technology (output devices, input parameters)
Penetration	1	Covers small part of the job
Inputs	1	Clients preferences
Ambiguity	2	Analyse clients preferences
Outputs	4	System can be used flexibly
Knowledge complexity	212	

Table 11.6: Knowledge complexity of a communication template application.

Breadth of the domain	1	Time registration rules
Depth of the domain	2	Degree in law or > 10 years experience
Change of the domain	3	Frequent changes in knowledge sources
Penetration	2	Single field of expertise required
Inputs	2	Labour law Sectoral agreements Labour contracts
Ambiguity	2	Use of jargon in inputs
Outputs	4	System can be used flexibly
Knowledge complexity	284	

Table 11.7: Knowledge complexity of a time registration system.

11.1.2 Comparing complexity scores

Although the knowledge complexity scores are not representative for the entire class of application types as we have defined them, they do give a feeling of how they probably relate to each other. Table 11.8 contains an overview of the knowledge complexity scores that we calculated in the previous paragraphs. From our limited sample of applications, it seems that the knowledge complexity is high in ETO PCSs compared to other applications. The main reason for this higher score is the penetration of expert knowledge. For the creation of a PCS, the product engineer needs to use different areas of expertise such as design knowledge, material knowledge, knowledge on production processes and production costs,... Consistent with this, product engineers need to take many inputs into account. In other application types, often only a part of the expert knowledge is used.

	Legal	ETO PCS	Scheduling	Collateral Selection	Railway Communication	Time Registration
K.C.Score	296	308-348	212	216	212	284
Percentage	70.48	73.33-82.86	50.48	51.43	50.48	67.62

Table 11.8: Overview of complexity scores.

If we maintain the mean score of 57 to distinguish high versus low complexity knowledge, we see that the scheduling application, collateral application and railway application

contain low complexity knowledge. The other applications contain high knowledge complexity.

11.1.3 Evaluation of the complexity measure

The method as described by Meyer and Culey has the advantage that it reveals some factors that very likely impact the complexity of a system. However, despite the fact that they have tried to describe meticulously how the different criteria should be measured, we found this to be a difficult exercise in practice. The concept of information unit for example, does not clearly describe how this unit should be interpreted. Therefore the scoring of a system using this measure, has a subjective component. This is demonstrated by the score range for PCSs for example, where we had difficulties deciding on an actual score. In the context of this dissertation, in which we assigned scores to the different cases alone, the knowledge complexity score probably is a good indication of the relative complexity of each case because the criteria are interpreted in the same way. However, care should be taken to appraise the ranking of scores as a truthful representation of the actual complexity ranking. Scoring by one analyst also means that subjective biases, such as the hypotheses that one kind of applications is more complex than the other kinds, might subconsciously influence the scoring process.

11.2 Developing an objective complexity measure

The complexity measure presented by Meyer and Culey is subjective in the sense that its criteria are open for interpretation. Moreover, the knowledge complexity that they investigate, is the complexity of the underlying knowledge of the expert who contributes to the system. We are looking for an objective measure of the complexity of a KB. To compare the complexity of one KB with the other, one could simply count the number of concepts in the vocabulary and the number of sentences in the theory. This is similar to the "lines of code" approach that is often used as complexity measure in software code. This measure primarily measures the size of the KB. Although larger programs tend to be more complex by their sheer size, more sentences in a KB might be the result of splitting long sentences in shorter once, hence improving readability instead of complexity. Ghosh et al. [56] developed a metric to measure the cognitive complexity of PCSs.[1] Their primary purpose is to be able to estimate the effort needed to develop and maintain a PCS, and not specifically the comparison of PCSs with other kind of systems. Their method is based on the UML model of the PCS, where the total complexity of the configuration model is expressed as a function of the complexities of the constituent classes. The complexity of each class is expressed as a function of the cognitive weight of its attributes and business

[1] The choice for a cognitive complexity measure stems from our knowledge acquisition viewpoint. We are interested in the complexity of the underlying knowledge, rather than the computational complexity, of which numerous measures for description logics exist, e.g., [**?**].

rules. In this, the business rules are considered as a single method with a single block of layered business control structures. The cognitive weights are estimated, based on psychological experiments to measure the cognitive complexity of basic control structures conducted by Wang [145]. The advantage of the approach of Ghosh is that it is an objectively observable measure. However, it requires a computer program in an object oriented programming language. It cannot be used to evaluate the complexity of our KB, that is not a computer program but a collection of domain information.

We are inspired by the work of Wang [145] and Ghosh [56] to estimate the cognitive complexity of a KB by decomposing it and assigning cognitive weights to each component. The cognitive complexity of the KB is the sum of the cognitive complexities of its components. As the number and complexity of concept and business rules have a mutual multiplier effect on the complexity, we keep a distinction between "attributes" and "business rules". Translated to an FO(·) KB this is the difference between the ontology of the domain (as represented in the vocabulary), and the rules and constraints that govern the interactions between the concepts of the ontology (i.e., the theory).

Cognitive complexity of the ontology Volume is an important criteria to measure complexity. A first naive approach is to assign a cognitive estimate of 1 to each concept (type, predicate, function) of the vocabulary (i.e., Score 1). This ignores the fact that certain concepts are easier to interpret than others. The number of different concepts is limited: either it is a type, relation or function. However, the number of arguments used in the concept, creates complexity. Therefore we slightly amend the estimates of cognitive complexity such that a score of one is assigned for every concept, plus an additional point for every additional argument on top of the first one (Score 2).

We demonstrate this with a small example of selecting the components for a laptop. In the first column of Table 11.9 the cognitive complexity of the vocabulary is calculated with the naive approach, i.e., 1 point per concept. The second column shows the amended approach, with an additional point for each argument on top of the first one.

Cognitive complexity of the theory The second part is to measure the complexity of the theory. The complexity measures of Wang and Ghosh are based on the decomposition of a computer program in basic control structures, such as an "if-then-else" structure [145, 56]. The basic control structures are divided in five categories, that each contain one, two or three different substructures. Each substructure has the same cognitive weight. In an FO(·) theory there are enormous possibilities to combine concepts and connectors in a single sentence. It is impossible to decompose them in basic control structures. In logic programs the use of parse trees is a way to decompose and visualise a logic sentence in a tree with nodes and arrows.

A sentence that could be part of a KB that describes the components of a computer, is to say that a computer has exactly one *Make*. One way to model this is to say that if a computer has make m1, and has also make m2, then m1 and m2 represent the

Vocabulary	Score 1	Score 2
type Computer isa nat	1	1
type Make constructed from {Hewlett, Lenovo, Apple, Asus}	1	1
type Keyboard constructed from {Azerty, Querty}	1	1
type Display isa nat	1	1
type RAM isa nat	1	1
type Storage isa nat	1	1
type Price isa nat	1	1
HasMake(Computer, Make)	1	2
HasKeyboard(Computer, Keyboard)	1	2
HasDisplay(Computer, Display)	1	2
HasRam(Computer, RAM)	1	2
HasStorage(Computer, Storage)	1	2
HasPrice(Computer, Price)	1	2
Cognitive complexity of vocabulary	13	19

Table 11.9: Knowledge complexity of a vocabulary.

same make:

$$\forall\ c[Computer]\ m1\ m2[Make] : HasMake(c, m1) \wedge HasMake(c, m2) \Rightarrow\ m1 = m2. \tag{11.1}$$

The simplified version of the parse tree (without tokens) of this sentence is given in Figure 11.1.

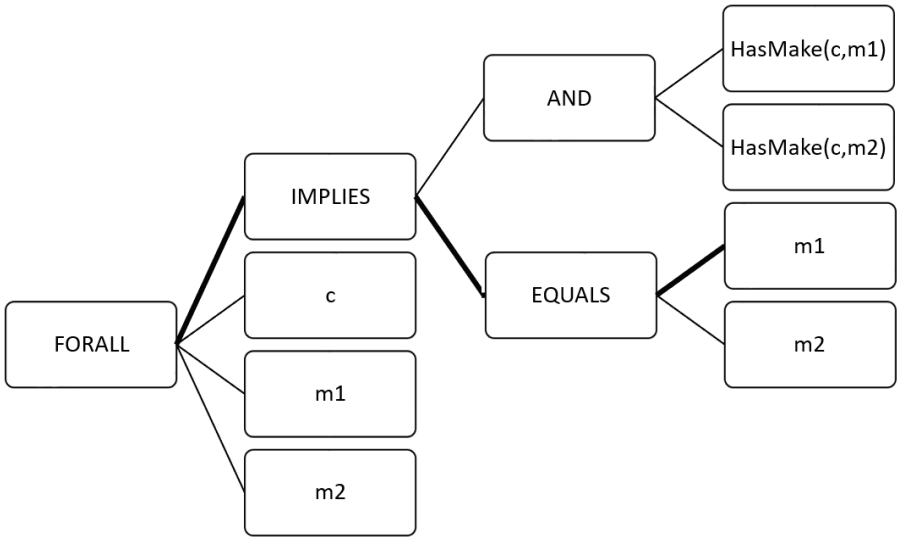


Figure 11.1: Parse tree of sentence 11.1 in FO(.)

If we return to the definition of complexity of a system as given by the volume of interdependencies and degree of interaction between the elements of a system, it suggests that especially the connection between elements drives the complexity. This would imply that a good way to calculate the complexity of a parse tree is to count the arrows between nodes. In our example the total score according to this measure, is ten. This way of calculating complexity disregards the structure of the sentence, as the number of arrows between n nodes is always $n - 1$. However, what makes sentences complex, is the nesting of different conditions. In a parse tree this results in a "deeper" tree, i.e., a tree in which the way from the top node to the deepest leave is longer. (Note that the tree is traditionally oriented vertically, but is shown horizontally here for better readability.) We adjust the complexity measure of a sentence by multiplying the number of arrows with the depth of the tree. The depth of the tree is defined as the longest path from the root of the tree to the deepest leave (the path in bold in Figure 11.1. The complexity measure for the tree in Figure 11.1 is 30 (10*3). In FO($\cdot$) also exists the possibility to use definitions. The calculation of their complexity requires a special approach, because a definition consists of multiple rules, yet each additional rule increases the complexity of the definition. So on top of the calculated complexity of each sentence, we also add the number of rules to the complexity measure.

11.3 Comparing complexity of KBs

As stated earlier, one of the purpose of developing a complexity measure for KBs, is to be able to compare complexity of different FO($\cdot$) models. In this section we compare the complexity of the registration duty prototype, the ETO product configuration prototype, and the doctor scheduling prototype. The third party applications are not part of this comparison as a part of the required domain knowledge is only available at the third party. A second reason to not include these applications in the comparison is that they were developed using IDP3, whereas the complexity measure functionality was developed for IDP-Z3.

For each KB there are four scores available: complexity score of the vocabulary, average complexity score of the constraints, average complexity score of the definitions, total complexity score of the theory, and total complexity score. The results for each application can be found in Table 11.10.

	Registration duty	Product configuration system	Scheduling application
Vocabulary	54	68	63
Avg. constraint	86.1	32.11	126.95
Avg. definition	205	320.5	30
Theory	1277	2181	2823
Total	1331	2249	2886

Table 11.10: Knowledge complexity of different applications.

From these numbers we can understand why one of the criteria of a qualitative scheduling system is that it should be able to deal with complex constraints. In the scheduling application we used very little, and simple, definitions. The constraints however, cover a wide range of situations: from hard constraints that are the same for every doctor, to habits that relate to a single doctor. These constraints basically use the same concepts, such as the planned shift, worked shift, night and weekend shifts. The complexity of the vocabulary is slightly lesser for scheduling applications than for PCSs. Unexpectedly the registration duty application is far less complex than the other applications. We suspect that this has more to do with the level of detail of the application, than with the inherent complexity of the domain. Both the scheduling and product configuration application contain very detailed requirements. For the registration duty application on the other hand, an important focus of the prototype was to demonstrate different use-cases of the system. Therefore the domain knowledge in the evaluated prototype has been scoped more narrowly then would be done for a real-life application. In casu, the portability of registration duties is not part of the KB. This impacts the score for vocabulary and theory. What is interesting to notice about the complexity measure, is that registration duty calculation displays a high average complexity for definitions. In the analysis of knowledge sources for legal applications, we observed that definitions are an important part of legal texts. The PCS also shows a high average for definitions. This is because the choice for a certain variant of a component is typically defined as a default choice and some specific situations in which other variants are preferred. The numbers from our complexity measure do not support the claim that PCSs are more complex systems. There might be several reasons why this is the case.

- To start with, it is possible that our measure does not correctly capture the complexity of the KB. The measure that we have developed is built on reasonable premises. However, its external validity needs to be confirmed by conducting experiments.
- Our cases might not be representative for the entire class of legal/PCS/scheduling applications. For example, our doctor planning case models the individual preferences of each doctor (next to the generic ones). Many scheduling applications only work with general constraints and preferences. Or, being prototypes, the scope of our prototypes may differ from the typical scope of a production application.
- The complexity of the KB according to this measure depends on the way the knowledge is modelled. One case might be better modelled than the others, leading to differences in the complexity score.
- Finally, PCSs might not be more complex than KBSs in other domains. Compared to imperative software, KBSs contain much more domain knowledge. This domain knowledge is also necessary for PCSs, regardless of the followed programming paradigm. Possibly imperatively programmed PCSs are more complex than many other imperative programs. But when KBSs are compared, this difference might disappear.

11.3.1 Other uses of the complexity measure

We have described above that different reasons may explain the different complexity scores of our applications. Consequently, users should take care to generalize the findings to entire categories of applications. The measure is more reliable if some of the alternative explanations can be ruled out. For example, it could be used to compare the complexity of two models of the same domain. When using the complexity measure to this end, three of the four cited explanations for difference in scores are addressed. The remaining reason, i.e., a different way of modelling, is exactly the reason to use the complexity measure. Indeed, modellers are typically concerned to write understandable models. Our complexity measure can assist them in choosing the model that most efficient from a knowledge complexity point of view.

11.4 Conclusion and future work

The use of a knowledge complexity measure for KBs allows to compare the complexity of different applications. There currently does not exist an objective measure to measure specifically the complexity of KBs. We propose a measure based on the decomposition approach of Wang [145] which consists of a score for the vocabulary, and one for the theory of a KB. The vocabulary is measured by assigning a point to every concept of the vocabulary, and for each concept adding points for every argument (as from the second) that is used in it. To measure the complexity of the theory, we use parse trees to decompose formulas. The complexity score of a formula is given by the number of arrows times the depth of the tree, given by the longest path from the top node to the deepest leave. Sentences in a definition are valued separately, and extra points depending on the number of sentences are added to reflect increased complexity for larger definitions.

Based on the complexity measure, the claim that PCS are more complex than other kinds of applications, cannot be supported. However, a variety of reasons can explain the unexpected result. One factor is the external validity of the measure itself. This will need to be confirmed experimentally in the line with the work of Wang [145].

Chapter 12

Conclusion

In the introduction we described the aim of this dissertation as gaining understanding on how one should proceed to create a new KBS. Throughout it we have analysed various kinds of applications and formulated recommendations for their development. In this chapter we bring these different recommendations together in an encompassing decision model that consists of three steps, as demonstrated in Figure 12.1

At the start of a project, the first question to be asked is if the project is a KBS project. KBSs can contribute a large added value in cases where expert knowledge is available, certainty and explainability of the outcome is important, the environment is changeable; or if the aim of the project is to gather and store expert knowledge. If the company decides that a KBS is the best solution for the problem at hand, the next step is to determine the type of their project. In this thesis we distinguish four main categories of applications: legal applications, product configuration systems, scheduling applications and third party applications. The last category is an overarching category of applications that are developed for the use by external third parties. In this category a distinction can be made between applications that are developed one time to be used by multiple clients, and applications that have a generic base structure, but that need to be tailored for each individual client. In the final step the recommended knowledge acquisition techniques for a specific project are derived based on its characteristics and application type.

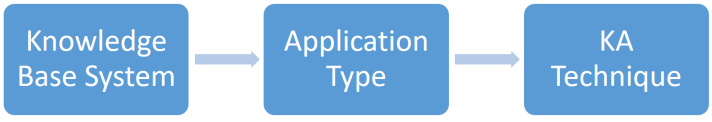


Figure 12.1: Main steps to decide on knowledge acquisition techniques.

12.1 Using a knowledge base system

The decision table below assists users in deciding if a KBS is appropriate for the kind of situation in which they want to deploy it.

Available knowledge (Expert, Data, Both) This refers to the access that the company has to a specific form of knowledge. Access to expert knowledge means that the company can obtain this knowledge from a human expert one way or the other. The typical example is an expert employee that can be interviewed to extract his domain knowledge. Of course, the expert does not need to be internal to the company, a consultant might also be able to reveal the necessary information. The expert knowledge can also be indirect, in the form of procedures, documents, contracts, existing applications, etc. If such indirect expert knowledge is used, it remains necessary to pass it by a human expert for validation and enrichment [28]. Note that the selection of experts is not a trivial task, and a lot of literature exists on how an expert stands out from more novice co-workers (e.g., [69]). Moreover, experts hold a certain status thanks to their unique knowledge, and are typically heavily solicited for it. Because of this, they might be reluctant to freely share their knowledge. Companies should take necessary actions to ensure that the experts have the time, possibility and willingness to collaborate. Access to data means that the company has its own data, has access to other data sources, or is able to synthetically create training data. It is possible that a company has not access to expert knowledge, nor to data for a specific application area. This probably means that the activity does not belong to the core activity of the company. Rather than creating an application for it, this activity can be outsourced or knowledge should be obtained one way or the other (collaboration with other companies, merger, training,...). Both means that the company has access to both expert knowledge and data.

Knowledge intensity (Low, Medium, High) This reflects the size of domain knowledge that needs to be included in the application.

Outcome (Certain, Estimate) An outcome is certain when it logically results from applicable rules. On a side note needs to be mentioned that the quality of the outcome depends on how complete and correct the rules in the application are. Some examples where outcomes need to be certain are contracts, law and internal procedures. If certainty is not absolutely required, one might be satisfied with an accurate enough estimate.

Environment (Dynamic, Static) If the environment is dynamic, the system should be able to cope with it, and respond correctly.

Explainable (Essential, Important, Less important) This reflects the importance of how well the end-user can obtain a detailed explanation on the obtained output.

Purpose (Knowledge harvesting, Knowledge discovery, Documentation, Training) Knowledge harvesting refers to the safeguarding of knowledge that is present in the company in employee's minds by representing it and making it available to others. Knowledge discovery is the creation of knowledge that was present but previously not accessible in the company. Documentation is the archiving of common knowledge in the company. Training is documentation with the specific purpose of educating novice employees.

KBS (HR, Possible, No, Hybrid) This column indicates whether a KBS can be used or not. "HR", "Possible" and "No" stands for highly recommended, possible to use a KBS and not possible to use a KBS respectively. The "Hybrid" indicates that a part of the application can be made of a KBS, but that a part of it uses other technologies.

12.2 Selecting an application type

This thesis covers four application categories: legal applications, product configuration systems, scheduling applications and third party applications. We are aware that this is a limited number of categories, and that many projects will not entirely fit in one of the predefined categories. The kind of knowledge from other projects may exhibit similarities with the knowledge in our applications, and some of the recommendations might be transferable. Therefore we created a decision table starting from the knowledge characteristics of the domain that allows to identify the application kinds most related to the project at hand.

Legal application A legal application was defined as an application to formalize statutory law and support decision making based on this statutory law. Characteristic for this kind of application is that the knowledge is available in multiple written sources. The information is structured in meaningful sections and paragraphs. It contains a lot of jargon and definitions. Although at first sight only deductive reasoning seems to be used, because of the uncertainty of events and partial available information, in practice default reasoning is mostly used by legal experts. The role of the expert is to interpret the written texts, and navigate through different sources of legislation. We suspect that the recommendations made for legal applications might also be applicable to applications that process written contracts because of their similar, highly-structured format and use of definitions. It is less clear if our conclusions can also be extended to business processes that are documented in written procedures. The fact that written documentation is available, is a source of similarity. A well-written procedure will also contain a clear structure. Different from law books, this structure is not standardised, and it is uncertain if a quality control of the text (that happens for legislation in

Knowledge Base System							
F	Available knowledge	Knowledge intensity	Outcome	Environment	Explainable	Purpose	KBS
1	Expert	not Low	Certain	–	–	–	HR
2	Expert	not Low	–	Dynamic	–	–	HR
3	Expert	not Low	–	–	Essential	–	HR
4	Expert	not Low	–	–	–	Knowledge harvesting	HR
5	Expert	not Low	–	–	–	Documentation	HR
6	Expert	not Low	–	–	–	Training	HR
7	Expert	not Low	Estimate	–	–	–	Possible
8	Data, Both	not Low	–	–	Essential	Knowledge discovery	Hybrid
9	–	–	–	–	–	–	No

Figure 12.2: Decision table on the use of a KBS

the form of approval by the legislative body) has happened. Laws and contracts will typically use a lot of definitions, to ensure a common understanding of the jargon they use. They do this with a legal precision to make the terms of the contract enforceable. In a company, employees probably also share a lot of jargon, but they might not even be aware of it. In that case it is not likely that the concepts will be explicitly defined in business documents. The recommendations for legal applications should be used with care in business procedure projects.

Product configuration system A system to support the design of products that consist of multiple parts in multiple variants by formulating constraints on how these parts can be combined to arrive at a product with the required characteristics. The product in question can be material or immaterial. Typically the circumstances in which the application is used, are subject to frequent changes due to technological progress, market conditions, regulatory changes, etc. The product experts occupy a very central role, and are typically well aware of developments in their field. Therefore they are best placed to adjust the KB. Often an explicit secondary concern in configuration projects is documenting the knowledge in the company. The possible design combinations are limited by the restrictions imposed by the company. This limits both the design freedom and complexity.

Scheduling application A scheduling application supports the creation of a work schedule, given a fixed number of shifts to be filled and a given number of available personnel. The main source of information is the planner, who can point towards legislation, company policy and employee's preferences that need to be reconciled in a fair schedule. The minimal requirements of the schedule are modelled in hard constraints, but often there are a great deal of preferences involved. These preferences are often hard to formalize and subject to change. Moreover, the planner occupies an important social role, that impacts the acceptance of the application. The role of time that is continuous in real life but divided in time periods in scheduling applications, is a specific point of concern. Very often a manual planning is created by trial and error, until a sufficiently good planning is obtained.

Third party application Third party applications are created by a company to support another company (the third party) in their decision process, where at least a part of the knowledge is only available at and specific for the third party. Although third party applications can be developed in any field, including legal, scheduling and product configuration fields, they are considered a separate category from a knowledge acquisition point of view. The fact that the necessary knowledge is not available to the developers of the application, is an essential element that influences the knowledge acquisition process. We make a distinction between third party applications tailored to the client, and generic third party applications. In the first kind, the ontology is adapted to the client's situation, which results in a highly specific application. In the latter kind, the ontology of the field is determined by the company, and the third party builds its own KB with the given ontology.

Based on the characteristics of each application type, we constructed the decision table in Figure 12.3 that directs users to the appropriate reference application type for their project, even if the domain is not the same. Note that the table is incomplete. It is part of future work to expand the number of case studies and describe more application types to cover all possible input combinations.

12.3 Choosing knowledge acquisition techniques

In the final step a (combination of) knowledge acquisition techniques is chosen based on the application type and the specific purpose of the application.

Legal application For legal applications the main sources of information are the legislation itself and legal experts who can provide an interpretation of the legislation, and navigate through the different applicable texts (e.g., regional, national and international legislation on one subject). To start modelling it is important to get insight in the structure of the applicable legislation. This can be obtained through interviews with a legal expert and analysis of the inherent structure of the legislative texts. A DMN DRD model is an excellent way of visualising this structure. As legal texts are themselves highly structured, using the principle of isomorphism results in a model that is easier to create for modellers and to understand for experts. The DRD can be used as high-level structure of which the decision blocks can be detailed one by one in decision tables. The role of the legal expert remains important throughout the entire process to provide interpretations of the text and validate the models that were created. The DRD can be used to structure interviews and refocus the discussion if it got digressed from. The legal domain is characterised by reasoning with exceptions. The use of first hit tables in this situation is suitable, as it results in tables that are shorter and better readable for legal experts than the default complete unique hit table. Although the (S)FEEL-based implementation tools for DMN do not offer enough flexibility with regards to reasoning with partial information, a DMN model is interesting as intermediate model to analyse the domain and model it without the need to focus on specificities of a complex modelling language. From the intermediate DMN model a FO($\cdot$) specification can easily be created. At first this may be used in a prototype to solicit the functional requirements of the system. As the use of computer systems by legal experts is typically rather restricted, a prototype serves well to allow experts to express their needs. Based on this feedback, and with the FO($\cdot$) specification as KB, the application can be developed step-by-step. Interactivity of the system, in which the conclusions of partial information are propagated, detecting relevant missing information and explanation of conclusions turn out to be important characteristics of a legal application.

Product configuration systems For PCSs a distinction is made between configurators that are intended to support MTO processes, and configurators that are intended to support ETO processes. A MTO configurator can be described as *deterministic*: during the configuration process the appropriate version of each component is chosen, resulting in the selection of one of the limited number of pre-defined designs. The best way to create a MTO PCS is by involving several experts in guided knowledge articulation workshops. Essential in it is the creation of one executable model that can be read, interpreted and adjusted by the domain experts themselves. In the supporting toolset that we developed, we use DMN as the unique modelling language. The knowledge articulation workshops start with the construction

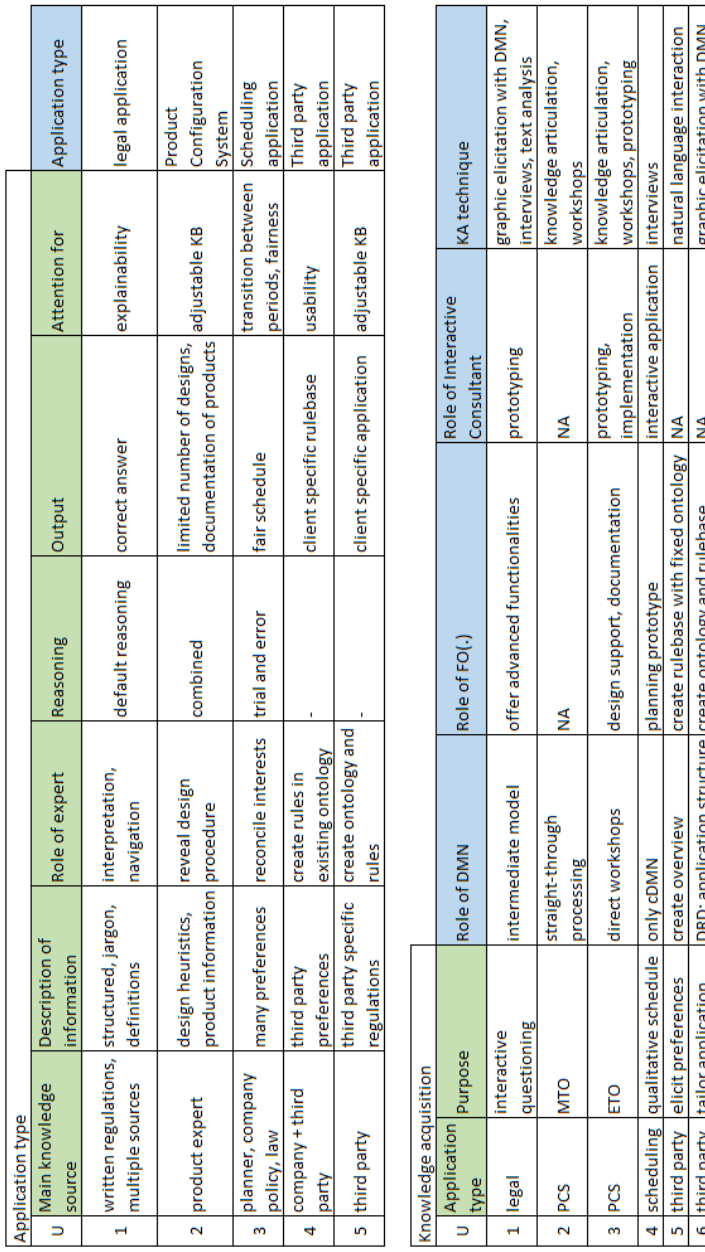
of a DRD of the design process, after which each decision is subsequently detailed in a decision table. The well-readable DMN model allows to leave unfinished decision tables as "homework" for experts that are endowed with missing pieces of knowledge and validation by experts who could not attend the workshops.

An ETO configurator is different from a MTO configurator in the sense that the design for the request of the client is not known yet. Instead of supporting a deterministic selection process, an ETO must support an unknown design process. In our case we started from the DMN model that we developed for the creation of the MTO configurator. We did this because the underlying reasons that explain the selection of specific parts in the MTO PCS are still valid in the ETO PCS. Not all companies will need to implement an MTO configurator. Nevertheless, the procedure of first creating a DMN model remains interesting to be able to structure workshops and ensure a structured approach of the design process. In doing so, the model acts as an intermediate model, similar to the model in legal applications. To extract the underlying knowledge of the DMN model, knowledge articulation workshops are organised. Each decision table is examined one-by-one to elicit the underlying reason(s) for its output. Most often, these reasons take the form of constraints. We developed a new modelling language, cDMN in which constraints can be modelled in the same easy-readable way as decisions. Moreover, the cDMN model can be automatically translated to the expressive FO($\cdot$) language. Besides this underlying domain knowledge, experts also know the reasoning process they follow to create a new design. The earlier obtained FO($\cdot$) model acts as KB for a prototype that is used to elicit the reasoning steps followed by the experts. It results in the development of additional features, such as the temporarily disregarding of a constraint and the browsing of historical designs database.

Scheduling applications Scheduling applications are a difficult kind of applications to develop. There are not only the technical challenges, such as dealing with the transition between subsequent planning periods and performance issues, but often planners engage in job crafting activities. These are often overlooked in automation projects. FO($\cdot$) is a relatively good modelling language for scheduling problems. It allows the formulation of hard and soft constraints that are applicable in the case study, and it carries most of the characteristics of a good planning system, such as the ability to easily add new constraints, maintainability and flexibility. We developed an interface that allows the planner to interactively play with the planning and its parameters (such as the weights assigned to soft constraints). However, the algorithmic power of the IDP system that we use is currently insufficient to support true interactivity. In the case we used a combination of interviewing and prototyping. The interviews were limited to the planner only. Because of the fairness objective and the aim to create a fair planning procedure, it would be recommended to involve more stakeholders, such as (a representation of) the employees. Prototyping is used in two ways: to validate the KB and elicit forgotten constraints, and to elicit functional requirements of the system. The latter is only applicable if the planner wants to interactively create the schedule.

Third party applications Some of the knowledge for third party applications is not accessible for the developer. We make a distinction between generic third party applications and client tailored applications. In the first kind of applications, the developing company has a good overview of the domain, but lacks knowledge of the preferences of the client. The company develops a single application with the defined ontology of the domain. Different clients can use this system with the pre-defined ontology to express their preferences. The use of a natural language interface proved to be successful to do this. The translation to FO(·) allows flexible functionalities of the system, and the implementation of supporting tools to keep overview of the created KB, such as a translation in decision tables. In the second kind of applications, that are client tailored, the developing company has access to some of the domain knowledge, and has developed a way to model and subsequently process this knowledge. However, depending on the client, the ontology used in the framework offered by the company, might not be sufficient. Clients are able to expand this, which results in a new, tailor made application. In this case, it is critical that consultants who implement the application at the client, have a good overview of the functioning of the application. This can be modelled in a cDMN model, in which the fixed parts cannot be altered by consultants.

In the right panel of Figure 12.3 the role of the different methods is summarised along with the recommended knowledge acquisition technique based on application type and purpose.



Application type

U	Main knowledge source	Description of information	Role of expert	Reasoning	Output	Attention for	Application type
1	written regulations, multiple sources	structured, jargon, definitions	interpretation, navigation	default reasoning	correct answer	explainability	legal application
2	product expert	design heuristics, product information	reveal design procedure	combined	limited number of designs, documentation of products	adjustable KB	Product Configuration System
3	planner, company policy, law	many preferences	reconcile interests	trial and error	fair schedule	transition between periods, fairness	Scheduling application
4	company + third party	third party preferences	create rules in existing ontology	-	client specific rulebase	usability	Third party application
5	third party	third party specific regulations	create ontology and rules	-	client specific application	adjustable KB	Third party application

Knowledge acquisition

U	Application type	Purpose	Role of DMN	Role of FO(.)	Role of Interactive Consultant	KA technique
1	legal	interactive questioning	intermediate model	offer advanced functionalities	prototyping	graphic elicitation with DMN, interviews, text analysis
2	PCS	MTO	straight-through processing	NA	NA	knowledge articulation, workshops
3	PCS	ETO	direct workshops	design support, documentation	prototyping, implementation	knowledge articulation, workshops, prototyping
4	scheduling	qualitative schedule	only cDMN	planning prototype	interactive application	interviews
5	third party	elicit preferences	create overview	create rulebase with fixed ontology	NA	natural language interaction
6	third party	tailor application	DRD: application structure	create ontology and rulebase	NA	graphic elicitation with DMN

Figure 12.3: Decision tables on application type (left) and knowledge acquisition technique (right).

12.4 Future work

12.4.1 General research directions

The cases that we have tackled in this work boiled down to four categories. One obvious way to expand it, is to **broaden the scope** to other cases. Doing other cases will not only lead to identifying new application types, it might also contribute to a better understanding of the currently identified application types. For example, a case on personnel planning might lead to an expansion of the current scheduling application type to planning and scheduling, but it might also result in a different category that gives insight on the difference between both fields.

In this thesis we have focussed on the creation of KBS to exploit expert knowledge. The combination of expert knowledge with other knowledge sources proves to be very successful to create real-life applications. For example in the PCS application that contains a historical design database, or the third party application that create a KB through a NL interface. Future work should focus on the **combination of expert knowledge with other knowledge sources** to facilitate the creation of a KB. An important knowledge source are natural language texts, such as company policies, legal texts, etc. In our registration duty case study this text was analysed and modelled in DMN manually, after which it was validated with the legal expert. Recent research has focused on automatic creation of a DRD from natural language texts [46], the automatic creation of decision tables [6], or a combination of both [113]. Besides future work on the principle of decision extraction to make it more powerful (e.g., by considering multiple sentences), from a knowledge acquisition point can be focussed on how to use this tool as part of a general knowledge acquisition process.

Another source of knowledge are existing artefacts, such as a design, planning, or profile. For the development of a new design, the access to the historical design database is an important advantage of the developed PCS. It builds on machine learning techniques to identify the historical designs to show to the user. Future work could exploit the knowledge inherent in the historical design database to learn undiscovered rules, and develop a pipeline to add them to the KB. Similar, in the scheduling application it was difficult to elicit all constraints and preferences. An analysis of past schedules could help to elicit tacit rules. In the case of investment profile creation, an existing relevant portfolio could help to discover rules to create an explicit investment profile.

In some cases, notably the scheduling and the time registration cases, the **performance** of the IDP system hampers the expansion in a production ready application. Future work should focus on improving performance. Various options can be investigated. It is known in personnel scheduling literature that the formulation of constraints impacts the performance. Hence, research can be done to adjust the FO($\cdot$) specification such that the used formulas are less resource intensive. Next, by imposing a solution mechanism in the KB the system could be directed to quicker find an acceptable solution. This has been demonstrated on the traveling salesmen problem

in clasp.[1] For this problem, the KB does not only contain the description of the problem domain, but it also contains the rule that from each node, the shortest path is chosen, assuming that this will end up in the shortest overall route. Research should demonstrate if this principle can be used for other types of applications. Moreover, dictating the system on how to find a solution goes against the principle of a declarative KB. A part of the research should answer the question if this is acceptable, and if yes, in which cases. Third, the cases described in this thesis use IDP3 or IDP-Z3 as solver. One apparent solution could be to use an other underlying solver. By using FOLASP the FO(·) specification can automatically be translated to ASP, which allows the comparison of a wide range of solvers [134].

12.4.2 Future work per application type

The generic research directions are not tied to a specific case, or have been observed in (almost) all cases. The following paragraphs describe future work to be executed in the context of a specific case.

Legal applications In testing the registration duty prototype our attention was drawn to the difference between what ought to be true according to the model, and what is true in reality. In the context of the application, Carbonnelle et al. solved the issue by introducing advice and situation variables [23]. In the application the behaviour of both variables is different. Advice variables are determined by the model. If they receive a propagated value, this creates the reality for that value. For situation variables on the other hand, a propagated value should be checked against its value in the environment. A propagated value is "intermediate" in the sense that it needs to be confirmed afterwards. In the interface this is reflected by propagating values in orange, and turning them green after manual confirmation. Imagine that we want to expand the application so it can also be used by the administration that levies the duty. Then an additional level of certainty needs to be added. If presented with a piece of information by the notary (e.g., an invoice of earlier paid registration duties), the administration will cross-check it against their own data to determine if they accept this proof or not. So, in the entire chain of registration duty calculation there are different ways to determine the value of a variable: propagation by the model, claim by the client, evidence by the client, and evidence accepted by the administration. This represents different levels of certainty. Future work should focus on the possible truth values of the assigned value, and the impact of it on the development of applications. The focus should be broader than only legal applications, and should be expanded to other types of applications as well.

In the case of registration duties we demonstrated that the use of first hit tables is convenient to reflect legal default reasoning. However, in general the use of first hit tables is strongly dissuaded, both in the DMN standard itself as in academic

[1] https://potassco.org/doc/videos/2016/09/20/optimization-in-clasp-3-via-traveling-salespersons.html

literature [9, 60]. The main reasons are that the usage of first hit tables would be more error-prone ([20]) and that they are harder to interpret and validate manually [9]. The findings presented in the case form an empirical basis for the arguments that first hit tables should be recommended in specific cases. Future research should investigate the arguments in favour and against the use of the first hit policy, and validate them by running experiments on validation times, execution times, construction times, etc. of different hit policies. Other ways to model default logic, such as the use of default values or output order, should be part of the comparison. As we demonstrated that first hit tables may have their advantages, future research could focus on combining different hit policies for the same table depending on the life-cycle phase of the table [92].

Product configuration systems We developed a metric to measure the complexity of KBs based on the metrics of Ghosh [56]. Ghosh based the cognitive weights in their model on experiments by Wang [145]. The measure we proposed is based on the same principle of decomposition in basic blocks. In object oriented programs these are basic control structures, in our KB it are the nodes and arrows of the parse trees of each sentence. We propose a score based on the number of nodes and depth of the tree, assuming that these are important factors of complexity. The assigned cognitive estimations should be validated experimentally as described in [145].

Scheduling applications We discussed that the IDP system with FO(·) as its KB input language has the potential to become a qualitative scheduling system. The factors that hamper this for the moment are the algorithmic power and lack of learning capabilities. Hence, future work should focus on these two elements.

We demonstrated in the scheduling application that the logic of the constraints can be modelled using cDMN. For the moment there is no adjusted DRD that allows to univocally model the decision structure in constraint based models. We have demonstrated the value of a visual decision structure in the knowledge acquisition process in multiple cases. Therefore this is an important area for development, and it is not as straightforward as one might expect to create an equivalent of a DRD for constraint reasoning. A DRD is a very readable representation of the hierarchical decision structure. Because of its simplicity, it offers the users a mental map of the domain. The DRD is not executable in itself. However, because the decisions in a DRD are linked to a specific piece of decision logic (in our case usually a decision table), it is often possible to create a one-on-one mapping. Some tools, such as Camunda[2], allow users of a DRD to zoom into the decision table. The DRD gives at the same time an overview of the input, intermediate and output variables. The inputs are explicitly added as oval inputs to the decisions. Intermediate variables are the outputs created by a decision table. If the table has only one output, it is recommended to name the table after the decision. Hence, the decision names are also the names of the intermediate variables. Following the same logic, the output

[2]https://docs.camunda.org/get-started/dmn/drg/

variable(s) is(are) the name(s) of the top decision(s). It is much more complex to represent the decision structure that can be represented by cDMN. To start with, the value of a variable can be impacted by more than one table. The constraint logic is not hierarchically structured and does contain loops between tables. There is no distinction between input, intermediate and output variables, and any variable from any input or output column can be a user given or calculated. An additional question is how the universal quantification can be represented. In the context of graphical representation of learning problems sometimes plates (i.e., rectangles) are used to represent the range of variables [17]. We could borrow this notation to represent universal quantification in cDMN, but the formal meaning still needs to be developed in this context. In cDMN specific structures, such as the nesting of quantifications, or quantification over two variables from the same type exist and should get their proper interpretation. So, it is not straightforward to come up with a diagram that is simple, can act as mental model of the decision structure, reflects all variables and can be mapped with the decision logic at the same time.

Train communication templates In this case we described how the Interactive Consultant can be used as PCS to add template rules to the KB. Future work will focus first on implementing the presented solution. Next, it needs to be investigated how the principle of adding rules to the KB can be broadened to rules in less strict settings without losing its user-friendliness.

Chapter 13

Valorization

This chapter focusses on the valorization of the research. Valorization is the process of commercialization of research results, and encompasses activities such as the creation of spin-off companies, patenting of academic research, technology transfer and academic entrepreneurschip [14]. Sometimes also broader activities, such as teaching, presenting the results to a business public and consultancy are considered to be part of the valorization process. A typical process to go from basic research to a commercial result, consists of nine technology readiness levels [47], that can be divided over three phases.

1. Discovery and research activities
   (a) Observation of basic principles
   (b) Formulation of a technology concept
2. Innovation activities
   (a) Development of an experimental proof of concept
   (b) Lab validation of the solution
   (c) Validation of (a part of) the solution in a relevant environment
   (d) Demonstration of a prototype in an operational environment
3. Commercialisation activities
   (a) Complete and qualified system that is tested in an operational environment
   (b) Actual system proven in operational environment

In phase one, the focus is on excellent science, in phase two it is on societal challenges, whereas in phase three the focus is on industrial leadership. These scales should not be considered as strictly linear steps towards the development of the end product.

Typically, during its development adjustments need to be made that require the revision of an earlier step. We have currently two prototypes (ETO PCS and collateral selection application) that have been validated in a relevant environment, and that are in the process of being implemented in an operational environment.

The basic research that underpins our research, is research on knowledge representation and reasoning (KRR) as developed at the Declaratieve Talen en Artificiële Intelligentie (DTAI) research group, and decision modelling as advanced by the Leuven Institute for Research on Information Systems (LIRIS) research group.

The first phase of my research was executed with the support of a Tetra-project that aims to promote the transfer of knowledge from research groups to an expansive target-group of Flemish companies. This was followed by a C3 project granted by KU Leuven to further develop results from the Tetra-project to support valorization of the results in a short time period. From this followed further contract research. As such, the case study descriptions of the previous chapters already contain a lot of information on the valorization of our results in the different case studies. Hence, in this chapter we will not elaborate on the valorization of each individual prototype. Instead we will summarize the most important factors that contributed to the valorization of the most successful cases, and extract the most important lessons that we learned from these valorization trajectories.

Subject of valorization Before diving into valorization paths and possibilities, we consider the valorization subject, or in other words, what can we offer? In the executed case studies we often used a combination of (c)DMN modelling, FO(·) modelling and prototyping based on the IDP system. The technologies, and the pipeline that was developed, were the central deliverables of our case studies. Next to the technology, we also developed considerable expertise on the development of KBSs. This includes knowledge acquisition methods to obtain domain knowledge, inference knowledge and task knowledge from experts. We also gained expertise in the valorization of our ideas.

Freedom to operate We constructed multiple prototypes to demonstrate the feasibility and interesting features of our approach and combination of technologies. Our prototypes have been developed in an academic environment. To commercialise the result, we need to ensure that we have the freedom to operate on the aforementioned technologies.

- DMN: The standard's specification is freely available online, as well as its normative and informative machine consumable files. The copyright companies that contributed to the development of the standard grant a "fully paid-up, non-exclusive, non-transferable, perpetual, worldwide license (without the right to sublicense)" to its users ([60], p.2). Users have the right to use the standard, and to develop and distribute software based on it. Moreover, the standard is deliberately vague at certain points, as to allow users to be creative in the development of their own implementation. So companies can use and build on

the DMN standard in commercial products. The only limitation to take into account is that the product may only claim to be compliant or conform with the standard if it is fully in line with the standard. If this is not the case, the product can only claim that it is based on the DMN standard.

- Z3: Z3 is the solver underlying IDP-Z3. It is created and owned by Microsoft. Users can obtain the software and its documentation freely on GitHub under an MIT license. This means that users have the right to use, copy, modify,... Z3 in their own (commercial) products. However, in case of damage caused by using Z3, Microsoft cannot be held responsible.
- IDP-Z3 and Interactive consultant: Developed at KU Leuven. The source code of IDP-Z3 is publicly available under the GNU LGPL v3 license, i.e., a general public license for free software. The components (including the interactive consultant) of IDP-Z3 are illustrated on the IDP-Z3 website, and the public is invited to contribute to its development.
- cDMN: Copyright by Simon Vandevelde, Bram Aerts, Marjolein Deryck and Joost Vennekens. The software is freely available under GNU General Public License v3.0.
- IDP: Copyright by DTAI. No explicit license granted.
- FO(·):Copyright by DTAI. No explicit license granted.
- MinisatID: This is the underlying solver of the IDP system. Copyright by DTAI (KRR. Available under GNU General Public License v3.0.

The technologies that we have used are open source and can be used in commercial applications.

Market position In the previous paragraph we investigate the freedom to operate the technology commercially. On the flip side there are currently no commercial entities that support users with the implementation and maintenance of the technology. This can either be seen as a challenge (because it is a threshold for companies willing to use our technology), or an opportunity (to create a spin-off that offers this service). If a spin-off company would be created to commercialise the technology and support it, the question arises if this company would be able to hold the market, or if other companies would quickly enter the market. Although the technologies are free to use, their development requires significant training in knowledge representation and reasoning. This represents an entrance barrier for new market participants. We derive from the positive evaluation of our prototypes that there is a market for our KBS solutions.

In the next sections we describe the most important elements from two successful valorization trajectories.

13.1 Product design

In Section 6 we described how we developed prototypes for PCSs. At the one hand we developed the DMN models for the MTO configurator, for which an OpenRules prototype was developed. The company further developed the prototype in a full OpenRules system, and integrated it with its existing systems. Important in this is that OpenRules is easy to develop and maintain by the company itself. In case of problems, OpenRules, Inc. can provide support.

On the other hand we developed an IDP based prototype for the ETO configurator. The prototype was evaluated positively, and the company wanted to move forward with the implementation. This is less straight-forward than the OpenRules implementation. To start with, additional development is required to move the prototype into a production ready version. Moreover, it is important for the company that the new PCS is maintained properly. Although the company developed in-house knowledge on the IDP system, this is too limited to offer full in-house support. Therefore the company needed to find an external partner to maintain and support the IDP system. As IDP is an academically developed system, it required quite some effort and negotiation to find a good partner. Moreover, the company was looking for additional funding for the development project, but got rejected. Both processes slowed down the implementation, which is currently back on track with a commercial partner to support the IDP system and contract research to assist with further development of the KB and inference tasks.

13.2 Asset selection

Early in the tetra project a basic prototype was developed to demonstrate the ability of the IDP system to select assets for investment profile eligibility. The case was initiated by Coppens and partners consulting, a small consulting agency that specialises in capital markets consulting. They used the IDP prototype to create a prototype of an application with a natural language interface. This improved prototype was presented to a large financial institution that subsequently launched a project to develop and implement the application together with Coppens and partners consulting. We were involved in this development in the form of contract research. The success of this valorization trajectory is the combination of different factors, that are linked to the collaboration with Coppens & partners consulting as industry experts:

- Identification of a compelling problem.
- Taking advantage of established connections and reputation when presenting the prototype.
- Offering a complete solution: KBS integrated in a natural language interface.
- Having a known and reliable partner to build a system to the needs of the client and maintain it.

13.3 Lessons learned from valorization trajectories

The sections above describe our most successful valorization trajectories. Besides these success stories there were also some successfully developed prototypes that did not (immediately) produce valorization results. From both successful and less successful valorization, valuable lessons can be learned.

- Collaboration with an expert in the industry to identify interesting areas for improvement that can be addressed by a KBS and establish connections with interested companies, is of utmost importance.
- We note that most IT departments are not acquainted with KBSs. This leads to reluctance to accept the new technology, and difficulties to maintain it if a project would be accepted. Therefore finding or establishing a company that can act as middle man to support this kind of technology and can assist in the implementation, is an important factor to promote further distribution of KBSs.
- Because of the threshold to enter a new technology, a solution should be able to answer different concerns of the users. An encompassing solution includes the creation and maintenance of the KB besides using it. This requires the combination of KBSs with other technologies.

13.4 Future valorization steps

Contract research has proven to be a good way to valorize our research results. Some of our prototypes are currently in the process of being implemented. The successful valorizations already taught us a lot about both or product and the valorization trajectory itself. On our product we found that it can offer a big added value, especially if it is combined with other technologies. This combination was not the focus of the current research. To develop an encompassing solution for companies, additional valorization research will need to be carried out. If we want to convince companies to enter into a research project this might result in contract research, we need to be able to show them what we can offer. Although our research group delivered many prototypes and published extensively about them, the professional communication is currently missing. Therefore the next step is to create an attractive website that shows our realizations. The target group of the website are Belgian companies and funding offices. The result of further valorization research will be on the one hand more mature prototypes that address several concern by combining different technologies, and on the other hand an extended network of companies. We learned from the successful valorization that the help of market experts to promote our technologies is indispensable. Moreover, by the end of the valorization research the prototypes that are currently being implemented, will be operational and might be used as success-stories in sales pitches. It will probably be a good moment to consider the creation of a spin-off company. The next steps, actions and foreseen results are summarised in Figure 13.1.

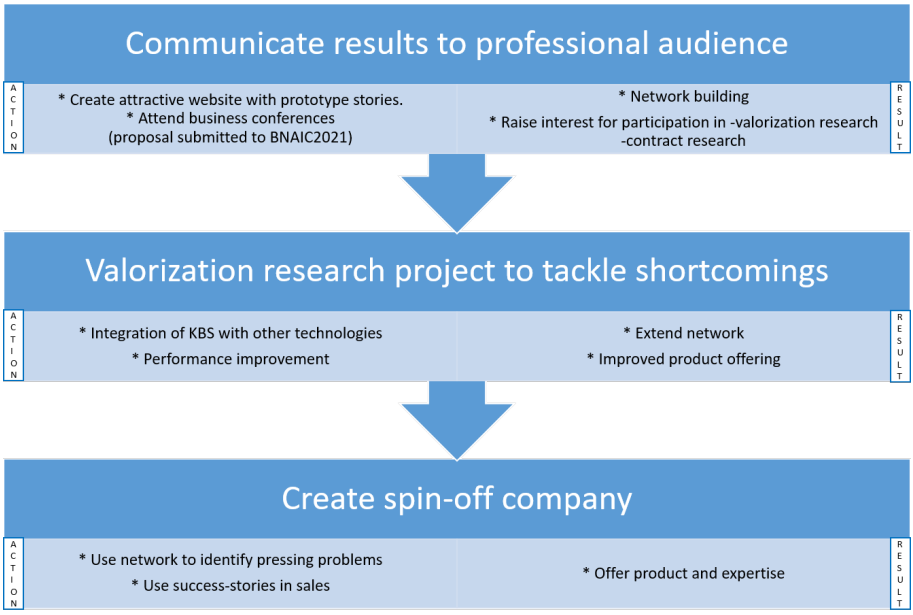


Figure 13.1: Next valorization steps.

Appendix A

Interview protocol

Interview protocol to interview planners in the context of scheduling applications, based on the protocol by [80]

1. How long have you been in this position?
   (a) Did you work another job in [facility/unit] before becoming the scheduler?
   (b) Do you have other tasks besides scheduling?
   (c) Who else do you work with to set the schedule? What role do they play?
2. First, let's talk about the overall or weekly schedule. In general, what is the process for setting the schedule in [facility/unit]?
   (a) How many employees need to be scheduled?
   (b) Which work regimes do you have in the [facility/unit]?
   (c) How much variation is there week-to-week?
   (d) Are there certain times of the year that are the most challenging for scheduling?
   (e) How do you manage these times?
   (f) Do you take employee's preferences into account?
      i. How do you know individual employee's preferences?
      ii. Can you list the different preferences for each employee?
      iii. How do you rank the importance of preferences?
      iv. Do these preferences stay the same throughout the year?
3. What is the biggest issue around schedules or work hours in [facility/unit]?
   (a) Think of the toughest scheduling problem you've ever faced on this job. Were you able to solve it? Can you tell me about that? What strategies did you use to try and solve it? (Probe: using floaters, per diems)

4. What if an employee needs a day off in advance? How far in advance do they need to put in a request?
    (a) Who approves the request?
    (b) Do you find coverage for them or is the employee responsible for that? (Probe: If scheduler finding coverage: What are the typical steps you go through to find coverage?)
    (c) Can you think of a time when a request was denied? Why was it denied? Is that typical?
5. Is the process the same for vacation requests, in terms of how far in advance the request needs to be turned in and approval? (If no, ask how it is different)
    (a) What is the policy at [facility/unit] regarding the amount of vacation that can be taken at one time?
    (b) Are exceptions ever made for this? can you give me an example?
6. We've talked about planned absences; now let's talk about call outs. How are last-minute requests for time off handled in [facility/unit]?
    (a) Does it matter if the reason is that the employee is ill as compared to other reasons, such as a sick child or other personal matter?
    (b) How do you go about finding coverage for call outs? Who do you go to first, then second, etc.?
    (c) Are there times when employees are responsible for finding their own coverage?
7. At what point is disciplinary action taken against someone for calling out too frequently? What action is taken?
    (a) Who is responsible for carrying out the disciplinary action?
    (b) Is this corporate policy or specific to [facility/unit]?
    (c) Can you think of a time when an exception to this policy was made?
8. To what degree are the policies regarding time off evenly enforced through [facility/unit]?
    (a) Are exceptions made for certain jobs or people? (ask for example or story)
9. Can you tell me about a time when you faced a conflict between labour costs, in other words staffing issues, and an employee's schedule preference? How did you manage that?
10. What is the official policy on overtime at [facility/unit]? Is this from corporate or from just [facility/unit]
    (a) Can you tell me about a time when you had to violate this policy? Did you get in trouble because of it?
11. How would an employee go about getting more hours on the schedule? What about reducing or cutting hours?

   (a) Again, does this approach differ at all between units in this facility? how?

12. What things would you like to change about how scheduling is handled?
13. Is there anything out of the ordinary that you yourself have tried out? Something you're particularly proud of?

Bibliography

[1] Adadi, A., and Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access 6* (2018), 52138–52160.

[2] Aerts, B. Advanced applications of knowledge representation technology, 2020.

[3] Aerts, B., Deryck, M., and Vennekens, J. Knowledge-Based Decision Support for Machine Component Design : a Case Study.

[4] Ahmed, S., Wallace, K. M., and Blessing, L. T. Understanding the differences between how novice and experienced designers approach design tasks. *Research in engineering design 14*, 1 (2003), 1–11.

[5] Aiken, L. H., Clarke, S. P., Sloane, D. M., Sochalski, J., and Silber, J. H. Hospital nurse staffing and patient mortality, nurse burnout, and job dissatisfaction. *JAMA - Journal of the American Medical Association 288*, 16 (2002), 1987–1993.

[6] Arco García, L., Nápoles, G., Vanhoenshoven, F., Lara, A., Casas Cardoso, G., and Vanhoof, K. Natural Language Techniques Supporting Decision Modelers. *Data Mining and Knowledge Discovery 35* (2021).

[7] Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., and Al, E. A framework for the development of personalized, distributed web-based configuration systems. *AI Magazine 24*, 3 (2003), 93–108.

[8] Barwise, J., and Etchemedy, J. *Language, proof and logic.* CSLI, Stanford, 2002.

[9] Batoulis, K., and Weske, M. Disambiguation of DMN Decision Tables. In *Business Information Systems* (Cham, 2018), W. Abramowicz and A. Paschke, Eds., Springer International Publishing, pp. 236–249.

[10] Ben Naylor, J., Naim, M. M., and Berry, D. Leagility: Integrating the lean and agile manufacturing paradigms in the total supply chain. *International Journal of Production Economics 62*, 1 (1999), 107–118.

[11] Bench-Capon, T. J. M., and Coenen, F. P. Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law 1*, 1 (1992), 65–86.

[12] Bennett, A. Interactive Aesthetics. *Design Issues 18*, 3 (jul 2002), 62–69.

[13] Boley, H., Zou, G., Mitsikas, T., Thom, M., Athan, T., and Chaudhari, R. PSOA RuleML, 2020.

[14] Bonaccorsi, A., Blasi, B., Nappi, C. A., and Romagnosi, S. Quality of research as source and signal: revisiting the valorization process beyond substitution vs complementarity. *The Journal of technology transfer* (2021).

[15] Böðvarsdóttir, E. B., Smet, P., Vanden Berghe, G., and Stidsen, T. J. R. Achieving compromise solutions in nurse rostering by using automatically estimated acceptance thresholds. *European Journal Of Operational Research 292*, 3 (2021), 1–16.

[16] Brebels, L. Mirror, mirror on the wall... Procedural fairness as an evaluative and regulatory looking-glass self, 2011.

[17] Buntine, W. L. Operations for Learning with Graphical Models. *CoRR abs/cs/941* (1994).

[18] Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F. M., Montali, M., and Teinemaa, I. Semantics and Analysis of {DMN} Decision Tables. In *International Conference on Business Process Management* (2016), Springer, pp. 217–233.

[19] Calvanese, D., Montali, M., Dumas, M., and Maggi, F. M. Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge. *Theory and Practice of Logic Programming 19*, 4 (2019), 536–573.

[20] Camunda Services GmbH. Choosing the DMN hit policy, 2021.

[21] Car, N. J. USING decision models to enable better irrigation Decision Support Systems. *Computers and Electronics in Agriculture 152* (2018), 290–301.

[22] Carbonnelle, P., Aerts, B., Deryck, M., Vennekens, J., and Denecker, M. An Interactive Consultant, 2019.

[23] Carbonnelle, P., Bogaerts, B., Vennekens, J., and Denecker, M. Interactive Advisor for Lax Legislation and Observable Situations, 2020.

[24] Chaves, A. P., and Gerosa, M. A. How Should My Chatbot Interact? A Survey on Social Characteristics in Human-Chatbot Interaction Design. *International journal of human-computer interaction 37*, 8 (2021), 729–758.

[25] Chen, J., and Hao, Y. Mass customization in design of service delivery system: A review and prospects. *African Journal of Business Management 4* (2010), 842–848.

[26] Chiaramonte, M. V., and Chiaramonte, L. M. An agent-based nurse rostering system under minimal staffing conditions. *International Journal of Production Economics 114* (2008), 697–713.

[27] Clark, K. L. *Negation as Failure*. Springer US, Boston, MA, 1978, pp. 293–322.

[28] Cooke, N. J. Varieties of knowledge elicitation techniques. *International Journal of Human-Computer Studies 41*, 6 (1994), 801–849.

[29] Crilly, N., Blackwell, A. F., and Clarkson, P. J. Graphic elicitation: using research diagrams as interview stimuli. *Qualitative Research 6*, 3 (2006), 341–366.

[30] Dasseville, I., Janssens, L., Janssens, G., Vanthienen, J., and Denecker, M. Combining DMN and the knowledge base paradigm for flexible decision enactment. In *Supplementary Proceedings of the RuleML 2016 Challenge, RuleML, Stony Brook, 6-9 July 2016* (2016), T. Athan, A. Giurca, R. Grütter, M. Proctor, K. Teymourian, and W. Van Woensel, Eds., CEUR-WS.org.

[31] de Cat, B., Bogaerts, B., Bruynooghe, M., and Denecker, M. Predicate Logic as a Modelling Language: The IDP System. *CoRR abs/1401.6* (2014).

[32] Denecker, M. Extending Classical Logic with Inductive Definitions. In *Computational Logic — CL 2000* (Berlin, Heidelberg, 2000), J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, Eds., Springer Berlin Heidelberg, pp. 703–717.

[33] Denecker, M. Building a knowledge base system for an integration of logic programming and classical logic. vol. 5366, Springer, pp. 71–76.

[34] Denecker, M., Brewka, G., and Strass, H. A formal theory of justifications. In *13th International Conference on Logic Programming and Non-monotonic Reasoning* (2015), Ianni, Giovambattista, Springer, pp. 250–264.

[35] Denecker, M., Pelov, N., and Bruynooghe, M. Ultimate Well-Founded and Stable Semantics for Logic Programs with Aggregates. In *Logic Programming* (Berlin, Heidelberg, 2001), P. Codognet, Ed., Springer Berlin Heidelberg, pp. 212–226.

[36] Denecker, M., and Vennekens, J. The well-founded semantics is the principle of inductive definition, revisited. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning* (2014), Chitta, Baral, AAAI Press, pp. 1–10.

[37] Deryck, M., Aerts, B., and Vennekens, J. Adding Constraint Tables to the DMN Standard: Preliminary Results. In *International Joint Conference on Rules and Reasoning* (2019), Springer, pp. 171–179.

[38] Deryck, M., Comenda, N., Coppens, B. J., and Vennekens, J. Using Logic to Decide on the Eligibility of Assets and Facilitate the Collateral Process, 2020.

[39] Deryck, M., Comenda, N., Coppens, B. J., and Vennekens, J. Combining Logic and Natural Language Processing to Support Investment Management. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning* (Hanoi, 2021), p. 5.

[40] Deryck, M., Devriendt, J., Marynissen, S., and Vennekens, J. Legislation in the knowledge base paradigm: interactive decision enactment for registration duties. In *proceedings of the 13 the IEEE Conference on Semantic Computing* (2019), IEEE, pp. 174–177.

[41] Deryck, M., Hasic, F., Vanthienen, J., and Vennekens, J. A Case-Based Inquiry into the Decision Model and Notation (DMN) and the Knowledge Base (KB) Paradigm. In *Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings* (2018), pp. 248–263.

[42] Deryck, M., Mitsikas, T., Almpani, S., Stefaneas, P., Frangos, P., Ouranos, I., Boley, H., and Vennekens, J. Aligning, Interoperating, and Co-executing Air Traffic Control Rules Across PSOA RuleML and IDP. In *Springer Lecture Notes in Computer Science* (2019), Springer.

[43] Deryck, M., and Vennekens, J. An Integrated Method for KnowledgeManagement in Product Configuration Projects. In *Springer Lecture Notes in Mechanical Engineering.* (Aalborg, 2021), p. 11.

[44] Duffy, A., and Andreasen, M. Enhancing the evolution of design science. 29–35.

[45] Eén, N., and Sörensson, N. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing* (Berlin, Heidelberg, 2004), E. Giunchiglia and A. Tacchella, Eds., Springer Berlin Heidelberg, pp. 502–518.

[46] Etikala, V., Vanthienen, J., and Van Veldhoven, Z. Text2Dec: Extracting Decision Dependencies from Natural Language Text for Automated DMN Decision Modelling. In *Lecture Notes in Business Information Processing* (2021), vol. 397, Springer Verlag (Germany), pp. 367–379.

[47] European Commission. Horizon 2020 Work programme 2014-2015 - General annexes revised. Tech. rep., 2014.

[48] Feldman, J. Representing and Solving Rule-Based Decision Models with Constraint Solvers, 2011.

[49] Ferreiros, J. The Road to Modern Logic-An Interpretation. *The Bulletin of Symbolic Logic 7*, 4 (2001), 441–484.

[50] Frishammar, J., Richtnér, A., Brattström, A., Magnusson, M., and Björk, J. Opportunities and challenges in the new innovation landscape: Implications for innovation auditing and innovation management. *European Management Journal 37*, 2 (2019), 151–164.

[51] Fu, J., Bastani, F. B., and Yen, I.-L. Model-Driven Prototyping Based Requirements Elicitation. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs* (Berlin, Heidelberg, 2008), B. Paech and C. Martell, Eds., Springer Berlin Heidelberg, pp. 43–61.

[52] Gaines, B. R., and Shaw, M. L. G. Eliciting knowledge and transferring it effectively to a knowledge-based system. *IEEE transactions on knowledge and data engineering 5*, 1 (1993), 4–14.

[53] García López, L. M., del Campo, D., Abellán Hernández, J., González, S., and Webb, L. A. Expert-novice differences in procedural knowledge in young soccer players from local to international level. *Journal of human sport and exercise 5*, 3 (2010), 444–452.

[54] Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S., and Wanko, P. Potassco User Guide. Tech. rep., University of Potsdam, Potsdam, 2019.

[55] Gelfond, M., and Lifschitz, V. The Stable Model Semantics for Logic Programming. In *Proceedings of International Logic Programming Conference*

and Symposium (1988), R. Kowalski, Bowen, and Kenneth, Eds., MIT Press, pp. 1070–1080.

[56] Ghosh, A., Kristjandottir, K., Hvam, L., and Vareilles, É. Measuring the Complexity of Product Configuration Systems. In *CEUR Workshop Proceedings* (2018), vol. 2220.

[57] Gilmore, J. F. Knowledge base systems in computer aided technology. In *The 23rd IEEE Conference on Decision and Control* (1984), IEEE, pp. 586–590.

[58] Grabowski, M., Massey, A. P., and Wallace, W. A. Focus groups as a group knowledge acquisition technique. *Knowledge Acquisition 4*, 4 (1992), 407–425.

[59] Gross, C. N., Brunner, J. O., and Blobner, M. Hospital physicians can't get no long-term satisfaction – an indicator for fairness in preference fulfillment on duty schedules. *Health Care Management Science 22*, 4 (2019), 691–708.

[60] Group Object Management. Decision Model and Notation, 2019.

[61] Hall, E., Gott, S., and Pokorny, R. A Prodecural Guide To Cognitive Task Analysis: The PARI Methodology. Tech. rep., Air Force Material Command, Books AFB, 1995.

[62] Hasić, F., Corea, C., Blatt, J., Delfmann, P., and Serral, E. A Tool for the Verification of Decision Model and Notation (DMN) Models. In *Research Challenges in Information Science*, Lecture Notes in Business Information Processing. Springer International Publishing, Cham, 2020, pp. 536–542.

[63] Hasic, F., and Vanthienen, J. Complexity Metrics for DMN Decision Models. *Computer Standards & Interfaces 65* (2019), 15–37.

[64] Haug, A., Shafiee, S., and Hvam, L. The costs and benefits of product configuration projects in engineer-to-order companies. *Computers in Industry 105* (2019), 133–142.

[65] Heiskala, M., Paloheimo, K.-S., and Tiihonen, J. Mass Customisation of Services: Benefits and Challenges of Configurable Services.

[66] Hoffman, R., and Lintern, G. Eliciting and Representing the Knowledge of Experts. *The Cambridge handbook of expertise and expert performance* (2006).

[67] Hoffman, R. R. *How Can Expertise be Defined? Implications of Research from Cognitive Psychology*. Palgrave Macmillan UK, London, 1998, pp. 81–100.

[68] Hoffman, R. R., Coffey, J. W., Carnot, M. J., and Novak, J. D. An Empirical Comparison of Methods for Eliciting and Modeling Expert Knowledge. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting 46*, 3 (2002), 482–486.

[69] Hoffman, R. R., Shadbolt, N. R., Burton, A. M., and Klein, G. Eliciting knowledge from experts: A methodological analysis. *Organizational Behavior and Human Decision Processes 62*, 2 (1995), 129–158.

[70] Hua, J. Study on Knowledge Acquisition Techniques. In *2008 Second International Symposium on Intelligent Information Technology Application* (dec 2008), vol. 1, pp. 181–185.

[71] Hvam, L., Kristjansdottir, K., Shafiee, S., Mortensen, N. H., and Herbert-Hansen, Z. N. L. The impact of applying product-modelling techniques in configurator projects. *International Journal of Production Research 57*, 14 (2019), 4435–4450.

[72] ICAO. Doc 4444-RAC/501, Procedures for Air Navigation Services - Rules of the Air and Air Traffic Services.

[73] ICON. Bottleneck vacancies 2015. Tech. rep., European Commission, Brussels, 2016.

[74] inc., R. RuleML home, 2021.

[75] Iqbal, T., and Suaib, M. Requirement Elicitation Technique: A Review Paper. *International Journal of Computer & Mathematical Sciences 3*, 9 (2014), 1–6.

[76] Kaba, A., and Ramaiah, C. K. Predicting knowledge creation through the use of knowledge acquisition tools and reading knowledge sources. *VINE Journal of Information and Knowledge Management Systems 50*, 3 (jan 2020), 531–551.

[77] Kellogg, D. L., and Walczak, S. Nurse scheduling: From academia to implementation or not? *Interfaces (Providence) 37*, 4 (2007), 353–369.

[78] Khanam, S. A., Liu, F., and Yi-Ping, P. C. Comprehensive structured knowledge base system construction with natural language presentation. *Human-centric Computing and Information Sciences 9*, 1 (jun 2019), 1–32.

[79] Komarudin, Guerry, M.-A., Smet, P., De Feyter, T., and Vanden Berghe, G. A two-phase heuristic and a lexicographic rule for improving fairness in personnel rostering. In *Proceedings of the 10th international conference on the practice and theory of automated timetabling* (2014), Ozcan, E, PATAT, pp. 292–308.

[80] Kossek, E. E., Piszczek, M. M., McAlpine, K. L., Hammer, L. B., and Burke, L. Filling the Holes: Work Schedulers As Job Crafters of Employment Practice in Long-Term Health Care. *ILR Review 69*, 4 (2016), 961–990.

[81] Kowalski, R. Predicate Logic as Programming Language. *Information Processing* (1974), 569–574.

[82] Kowalski, R., and Sergot, M. A logic-based calculus of events. *New Generation Computing 4*, 1 (1986), 67–95.

[83] Kowalski, R. A., and Van Emden, M. H. *The semantics of predicate logic as a programming language*. University of Edinburgh. Department of computational logic. Memo 73. University of Edinburgh. Department of artificial intelligence, Edinburgh, 1974.

[84] Kristjansdottir, K., Shafiee, S., Hvam, L., Forza, C., and Mortensen, N. H. The main challenges for manufacturing companies in implementing and utilizing configurators. *Computers in Industry 100* (2018), 196–211.

[85] KU Leuven Knowledge Representation and Reasoning research group. The IDP framework reference manual. Tech. rep., 2020.

[86] Lang, S., Tittsworth, J., Bryant, W., Wilson, P., Lepadatu, C., Delisi, D., Lai, D., and Greene, G. Progress on an ICAO Wake Turbulence Re-Categorization Effort. *AIAA Atmospheric and Space Environments Conference* (2010).

[87] Lawsky, S. B. A Logic for Statutes. *Florida tax review 21*, 1 (2017), 60–80.

[88] Leu, G., and Abbass, H. A multi-disciplinary review of knowledge acquisition methods: From human to autonomous eliciting agents. *Knowledge-Based Systems 105* (2016), 1–22.

[89] Leventhal, G. S. *What Should Be Done with Equity Theory?* Springer US, Boston, MA, 1980, pp. 27–55.

[90] Liebowitz, J., and Megbolugbe, I. A set of frameworks to aid the project manager in conceptualizing and implementing knowledge management initiatives. *International Journal of Project Management 21*, 3 (2003), 189–198.

[91] Liu, W., Mo, Y., Yang, Y., and Ye, Z. Decision model of customer order decoupling point on multiple customer demands in logistics service supply chain. *Production Planning & Control 26*, 3 (2015), 178–202.

[92] Maes, R., and Van Dijk, J. E. M. On the Role of Ambiguity and Incompleteness in the Design of Decision Tables and Rule-Based Systems. *The Computer Journal 31*, 6 (jan 1988), 481–489.

[93] Maleki, A., Rosiello, A., and Wield, D. The effect of the dynamics of knowledge base complexity on Schumpeterian patterns of innovation: the upstream petroleum industry. *R&D Management 48*, 4 (2018), 379–393.

[94] Manson, S. M. Simplifying complexity: a review of complexity theory. *Geoforum 32*, 3 (2001), 405–414.

[95] Martin, S., Ouelhadj, D., Smet, P., Vanden Berghe, G., and Özcan, E. Cooperative search for fair nurse rosters. *Expert Systems with Applications 40* (2013), 6674–6683.

[96] McCarthy, J. Elaboration tolerance. *Common sense*, Symposium on Logical Formalizations of Commonsense Reasoning (1998), 24.

[97] McGraw, Karen L and Harbison-Briggs, K. *Knowledge acquisition: Principles and guidelines.* Prentice-Hall, Inc., 1989.

[98] Mertens, L., and Vennekens, J. Predicting Subcontractor Performance From Past Audits. In *2020 IEEE E-TEMS Conference Proceedings* (2020), IEEE; 2020 IEEE European Technology and Engineering Management Summit (E-TEMS).

[99] Mertens, S., Gailly, F., and Poels, G. Enhancing declarative process models with DMN decision logic. In *International Conference on Enterprise, Business-Process and Information Systems Modeling* (2015), Springer, pp. 151–165.

[100] Meyer, M. H., and Curley, K. F. An Applied Framework for Classifying the Complexity of Knowledge-Based Systems. *MIS Quarterly 15*, 4 (sep 1991), 455–472.

[101] MILITELLO, L. G., AND HUTTON, R. J. B. Applied cognitive task analysis (ACTA): a practitioner's toolkit for understanding cognitive task demands. *Ergonomics 41*, 11 (1998), 1618–1641.

[102] MOHUN, J., AND ROBERTS, A. Cracking the code.

[103] MOTIK, B., AND ROSATI, R. Reconciling description logics and rules. *Journal of the ACM 57*, 5 (2010), 1–62.

[104] NEALE, I. First generation expert systems: a review of knowledge acquisition methodologies. *Knowl. Eng. Rev. 3* (1988), 105–145.

[105] NURMI, K., KYNGÄS, J., AND POST, G. DRIVER ROSTERING FOR A FINNISH BUS TRANSPORTATION COMPANY. pp. 15–35.

[106] OECD. The knowledge-based economy. Tech. rep., Organisation for economic co-operation and development, Paris, 1996.

[107] OECD. Digital Government Index.

[108] OVERHEID, V. Decreet van 18 mei 2018 houdende wijziging van de Vlaamse Codex Fiscaliteit van 13 december 2013, wat betreft de hervorming van het verkooprecht en vereenvoudigingen in de registratiebelasting. *Belgisch Staatsblad*, 44142.

[109] PETROVIC, S. "You have to get wet to learn how to swim" applied to bridging the gap between research into personnel scheduling and its implementation in practice. *Annals of Operations Research 275*, 1 (2019), 161–179.

[110] PETROVIC, S., AND VANDEN BERGHE, G. A comparison of two approaches to nurse rostering problems. *Annals of operations research 194*, 1 (2012), 365–384.

[111] PIYALI SYAM. What is the Difference Between Common Law and Civil Law?, 2014.

[112] PUTTEMANS, S. Exploiting scene constraints to improve object detection algorithms for industrial applications, 2017.

[113] QUISHPI, L., CARMONA, J., AND PADRÓ, L. Extracting Decision Models from Textual Descriptions of Processes. In *Business Process Management* (Cham, 2021), A. Polyvyanyy, M. T. Wynn, A. Van Looy, and M. Reichert, Eds., Springer International Publishing, pp. 85–102.

[114] RAAD VAN STATE. Beginselen van de wetgevingstechniek, 2008.

[115] ROMER, P. Endogenous Technological Change. NBER Working Papers 3210, National Bureau of Economic Research, Inc, 1989.

[116] SCHÄFFER, E., SHAFIEE, S., MAYR, A., AND FRANKE, J. A strategic approach to improve the development of use-oriented knowledge-based engineering configurators (KBEC). *Procedia CIRP 96* (2021), 219–224.

[117] SCHUBERT CHRISTIANE C., M. S. P., DENMARK T. KENT, M. D., CRANDALL BETH, B. S., GROME ANNA, M. S., AND PAPPAS JAMES, M. D. M. B. A. Characterizing Novice-Expert Differences in Macrocognition: An Exploratory Study of Cognitive Work in the Emergency Department. *Annals of emergency medicine 61*, 1 (2012), 96–109.

[118] Schuwer, R. V., and Kusters, R. J. Application areas and added value of knowledge base systems. *Information & Management 24*, 2 (1993), 83–92.

[119] Seipel, D., Nogatz, F., and Abreu, S. Domain-specific languages in Prolog for declarative expert knowledge in rules and ontologies. *Computer languages, systems & structures 51* (2018), 102–117.

[120] Shafiee, S., Hvam, L., and Haug, A. Comparing the gained benefits from product configuration systems based on maintenance efforts. In *Proceedings of the 21th International Configuration Workshop* (2019), L. Hotz, M. Aldanondo, and T. Krebs, Eds., CEUR Workshop Proceedings, University of Hamburg, pp. 48–52.

[121] Shafiee, S., Kristjansdottir, K., Hvam, L., and Forza, C. How to scope configuration projects and manage the knowledge they require. *Journal of knowledge management 22*, 5 (2018), 982–1014.

[122] Shafiee, S., Wautelet, Y., Hvam, L., Sandrin, E., and Forza, C. Scrum versus Rational Unified Process in facing the main challenges of product configuration systems development. *Journal of Systems and Software 170* (2020), 110732.

[123] Simon, H. A. The Architecture of Complexity. *Proceedings of the American Philosophical Society 106*, 6 (jun 1962), 467–482.

[124] Smet, P. Nurse rostering: models and algorithms for theory, practice and integration with other problems, 2015.

[125] St Pierre, M., and Nyce, J. M. How novice and expert anaesthetists understand expertise in anaesthesia: a qualitative study. *BMC medical education 20*, 1 (2020), 262.

[126] Stolletz, R., and Brunner, J. O. Fair optimization of fortnightly physician schedules with flexible shifts. *European Journal of Operational Research 219*, 3 (2012), 622–629.

[127] Svarc, J., and Dabic, M. Evolution of the Knowledge Economy: a Historical Perspective with an Application to the Case of Europe. *Journal of the Knowledge Economy 8*, 1 (mar 2017), 159–176.

[128] Taylor, J., Fish, A., Vanthienen, J., and Vincent, P. Emerging standards in decision modeling. In *Intelligent BPM Systems: Impact and Opportunity* (2013), BPM and Workflow Handbook series, iBPMS Expo, pp. 133–146.

[129] Trunomi. GDPR Portal. \url{https://www.eugdpr.org/}.

[130] Tsarkov, D., and Horrocks, I. FaCT++ Description Logic Reasoner: System Description. In *Automated Reasoning* (Berlin, Heidelberg, 2006), U. Furbach and N. Shankar, Eds., Springer Berlin Heidelberg, pp. 292–297.

[131] Turan, A., Tunç, A. Ö., and Zehir, C. A Theoretical Model Proposal: Personal Innovativeness and User Involvement as Antecedents of Unified Theory of Acceptance and Use of Technology. *Procedia - Social and Behavioral Sciences 210* (2015), 43–51.

[132] Van Daele, D., Decleyre, N., Dubois, H., and Meert, W. An Automated Engineering Assistant: Learning Parsers for Technical Drawings. *arXiv preprint arXiv:1909.08552* (2019).

[133] van den Bergh, J., Belien, J., De Bruecker, P., Demeulemeester, E., and De Boeck, L. Personnel scheduling: a literature review. *European Journal of Operational Research 226*, 3 (2013), 367–385.

[134] Van Dessel, K., Devriendt, J., and Vennekens, J. FOLASP: FO(·) as Input Language for Answer Set Solvers. *Theory And Practice Of Logic Programming* (2021).

[135] Van Gelder, A., Ross, K. A., and Schlipf, J. S. The Well-Founded Semantics for General Logic Programs. *J. ACM 38*, 3 (jul 1991), 619–649.

[136] Van Hertum, P., Dasseville, I., Janssens, G., and Denecker, M. The KB paradigm and its application to interactive configuration. *Theory and Practice of Logic Programming 17*, 1 (2017), 91–117.

[137] Vandevelde, S., Aerts, B., and Vennekens, J. Tackling the DM Challenges with cDMN: A Tight Integration of DMN and Constraint Reasoning. *Theory And Practice Of Logic Programming* (2021).

[138] Vandevelde, S., Callewaert, B., and Vennekens, J. Context-Aware verification of DMN. In *Proceedings of the 55th Hawaii International Conference on System Sciences* (2021).

[139] Vanthienen, J., and Dries, E. Developments in decision tables: evolution, applications and a proposed standard, 1992.

[140] VDAB. Knelpuntberoepen in Vlaanderen 2021. Tech. rep., VDAB, 2021.

[141] Venkatesh, V., Morris, M. G., Davis, G. B., and Davis, F. D. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly 27*, 3 (dec 2003), 425–478.

[142] Vennekens, J., Denecker, M., and Bruynooghe, M. FO(ID) as an extension of DL with rules. *Annals of mathematics and artificial intelligence 58*, 1-2 (2010), 85–115.

[143] Vlaeminck, H., Vennekens, J., and Denecker, M. A logical framework for configuration software. In *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming PPDP '09, Principles and Practice of Declarative Programming, Coimbra, Portugal, 7-9 September, 2009* (sep 2009), ACM.

[144] Wagner, W. P. Issues in Knowledge Acquisition. In *Proceedings of the 1990 ACM SIGBDP Conference on Trends and Directions in Expert Systems* (New York, NY, USA, 1990), SIGBDP '90, Association for Computing Machinery, pp. 247–261.

[145] Wang, Y. Cognitive Complexity of Software and its Measurement. In *2006 5th IEEE International Conference on Cognitive Informatics* (2006), vol. 1, pp. 226–235.

[146] Wang, Y., Mo, D. Y., and Tseng, M. M. Relative preference-based product configurator design. *Procedia CIRP 83* (2019), 575–578.

[147] Wei, J., and Salvendy, G. The cognitive task analysis methods for job and task design: review and reappraisal. *Behaviour & Information Technology 23*, 4 (2004), 273–299.

[148] Wittocx, J., De Cat, B., and Denecker, M. Towards computing revised models for FO theories. In *Proceedings of the International Conference on Applications of Declarative Programming and Knowledge Management 2009, International Conference on Applications of Declarative Programming and Knowledge Management, Evora, Portugal, 5-7 November 2009* (nov 2009), S. Abreu and D. Seipel, Eds., pp. 199–211.

[149] Wittocx, J., Mariën, M., and Denecker, M. The idp system: a model expansion system for an extension of classical logic. In *Proceedings of the 2nd Workshop on Logic and Search* (2008), ACCO; Leuven, pp. 153–165.

[150] Yin, R. K. *Case study research: design and methods*, 5th ed. ed. Sage, Thousand Oaks, 2014.

[151] Zhang, L. L. Product configuration: A review of the state-of-the-art and future research. *International Journal of Production Research 52*, 21 (2014), 6381–6398.

[152] Zhao, H., McLoughlin, L., Adzhiev, V., and Pasko, A. "Why do we not buy mass customised products?" – An investigation of consumer purchase intention of mass customised products. *International Journal of Industrial Engineering and Management 10*, 2 (2019), 181–190.

[153] Zimmermann, M. The Universal Problem Solver – "AI Made in Potsdam" on its Triumphal Procession around the World, 2019.

Curriculum Marjolein Deryck

EDUCATION

KU Leuven *2017 - 2021*
Ph.D. Engineering Technology
Research groups DTAI, EAVISE
Thesis: Knowledge Base Systems in Practice:
Approaches, Application Areas and Limitations

Universiteit Antwerpen *2014 - 2016*
M.Sc. Business Engineering: Great Distinction
Management Information Systems
Thesis: Application of enterprise engineering concepts
within the financial sector: a descriptive case study

Universiteit Antwerpen *2012 - 2015*
B.Sc. Business Engineering: Distinction
Management Information Systems

KU Leuven *2002 - 2003*
Graduate in supplementary studies in Business Economics Distinction

KU Leuven *2000 - 2002*
Licentiate in Political Sciences Distinction

Universiteit Antwerpen *1998 - 2000*
Candidate in Communication Sciences

WORK EXPERIENCE

KU Leuven *2017 - 2021*
Researcher EAVISE, DTAI
Supervision by Joost Vennekens
Tetra project "Decision Analytics"
Promotors: Joost Vennekens, Marc Denecker, Jan Vanthienen

KU Leuven C3 Valorization project
Promotors: Joost Vennekens, Jan Vanthienen

National Bank of Belgium
Het combineren van kennisgebaseerde AI en beslissingsmodellen voor interbancaire transacties.
Promotors: Joost Vennekens, Jan Vanthienen

University of Antwerp *2016 - 2017*
Assistant Faculty of Business and Economics
Department Accountancy and Finance
Assisting in: Bachelor's theses, Investment Analysis, Banking, Financial Reporting and Analysis (AMS).

Belfius Bank *2007 - 2016*
Back-Office Financial Markets
Project manager
Teamcoordinator Accounting Financial Markets
Senior employee Accounting Financial Markets

Euroclear Bank *2004 - 2006*
Analyst Settlement
New Issues Distribution
Analyst Clearstream & Realignments

List of publications

Journal articles

[1] Aerts, B., Deryck, M., and Vennekens, J. Knowledge-Based Decision Support for Machine Component Design : a Case Study. *Expert Systems with Applications* (2021), 52.

(Accepted)

Conference publications

[1] Deryck, M., Aerts, B., and Vennekens, J. Adding Constraint Tables to the DMN Standard: Preliminary Results. In *International Joint Conference on Rules and Reasoning* (2019), Springer, pp. 171–179.

[2] Deryck, M., Comenda, N., Coppens, B. J., and Vennekens, J. Combining Logic and Natural Language Processing to Support Investment Management. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning* (Hanoi, 2021), p. 5.

[3] Deryck, M., Devriendt, J., Marynissen, S., and Vennekens, J. Legislation in the knowledge base paradigm: interactive decision enactment for registration duties. In *proceedings of the 13 the IEEE Conference on Semantic Computing* (2019), IEEE, pp. 174–177.

[4] Deryck, M., Dvorak, O., De Bruyn, P., and Verelst, J. Investigating the Evolvability of Financial Domain Models. In *ADVANCES IN ENTERPRISE ENGINEERING XI* (2017), vol. 284, Aveiro, D, Springer Verlag (Germany), pp. 111–125.

[5] Deryck, M., Hasic, F., Vanthienen, J., and Vennekens, J. A Case-Based Inquiry into the Decision Model and Notation {(DMN)} and the Knowledge Base {(KB)} Paradigm. In *Rules and Reasoning - Second International Joint*

Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings (2018), pp. 248–263.

[6] Deryck, M., and Huysmans, P. Application of Enterprise Engineering to Lean Process Management: An Explorative Case Study. In *ADVANCES IN ENTERPRISE ENGINEERING XI* (2017), vol. 284, Aveiro, D, Springer Verlag (Germany), pp. 141–148.

[7] Deryck, M., Mitsikas, T., Almpani, S., Stefaneas, P., Frangos, P., Ouranos, I., Boley, H., and Vennekens, J. Aligning, Interoperating, and Co-executing Air Traffic Control Rules Across PSOA RuleML and IDP. In *Springer Lecture Notes in Computer Science* (2019), Springer.

[8] Deryck, M., and Vennekens, J. An Integrated Method for KnowledgeManagement in Product Configuration Projects. In *Springer Lecture Notes in Mechanical Engineering.* (Aalborg, 2021), p. 11.

Abstract publications

[1] Deryck, M., Comenda, N., Coppens, B. J., and Vennekens, J. Using Logic to Decide on the Eligibility of Assets and Facilitate the Collateral Process, 2020.

[2] Deryck, M., Comenda, N., Coppens, B. J., and Vennekens, J. Combining Logic and Natural LanguageProcessing to Support Investment Management. In *BNAIC* (Esch-sur-alzette, 2021), p. 3.

Demos

[1] Carbonnelle, P., Aerts, B., Deryck, M., Vennekens, J., and Denecker, M. An Interactive Consultant, 2019.

[2] Deryck, M., Comenda, N., Coppens, B. J., and Vennekens, J. Logical Reasoning application with NLP interface to construct the Knowledge Base. p. 3.

Professional conferences

[1] Decleyre, N., Deryck, M., and Aerts, B. Combining DMN, First Order Logic and Machine Learning: The creation of Saint-Gobain Seals' Digital Engineer.

[2] Deryck, M. Decision Management with DMN. In *The Banking Scene* (2018).

[3] Deryck, M. Domeinexperten aan het stuur van automatisatie: een case study in decision modelling. In *Inspiratiedag over artificiële intelligentie van de Vlaamse overheid* (2018).

[4] Deryck, M. Meeting the Expectations with DMN and Constraint Solving: The Notary Case. In *Decision Camp* (2019), p. 1.



FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
EAVISE
Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne Waver
B-3001 Leuven
first.last@cs.kuleuven.be
https://iiw.kuleuven.be/onderzoek/eavise/index.html