Knowledge-Based Decision Support for Machine Component Design: a Case Study

Bram Aerts*, Marjolein Deryck, Joost Vennekens*

KU Leuven, De Nayer Campus, Dept. of Computer Science
Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium
Leuven.AI – KU Leuven Institute for AI, 3000 Leuven, Belgium

*Corresponding author

Email addresses: B.Aerts@kuleuven.be (Bram Aerts), Marjolein.Deryck@kuleuven.be (Marjolein Deryck), Joost.Vennekens@kuleuven.be (Joost Vennekens)

Abstract

This paper describes research conducted during a project with a multinational company that focuses on product design. The project tackles two different goals: providing sales staff with a tool that allows them to autonomously handle routine requests, and providing the company's engineers with a decision support system to help them design products for more challenging application areas. For the first goal, we make use of a deterministic decision process, represented in the recent Decision Model and Notation (DMN) standard. For the second goal, we propose a constraint-based method. There, we use the IDP system to provide a number of interactive functionalities based on a logical representation of the relevant constraints. To ensure that the system is maintainable, we want the constraints to be updated by the engineers themselves. The IDP language is not ideal for this. Instead, we propose the cDMN notation, which extends the user-friendly DMN to constraints.

Keywords: Deterministic modelling, DMN, Constraint-based reasoning, Interactive configuration

1. Introduction

This paper describes the design, implementation and evaluation of a knowledge base system 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 different materials. 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 (Wang et al., 2019). The focus of the application 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 situation was as follows.

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

Non-standard Designs. Requests that fell outside the scope of standard designs 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. For this kind of demands, the company wanted to standardize the design, and automate the process.

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. The company wanted to start providing such support.

Design Procedure. Previously, to design non-standard products, the engineers followed an *ad hoc* process. This process is highly individual, and is the result of the engineer's past experience, preferences, discussion with colleagues, etc. Although this way of working is still common in industry, it has several downsides. First, there is a lack of standardisation. This meant that different engineers may come up with different designs for the same set of requirements. It implies higher production costs and complexity in terms of product variants. Second, because of the lack of standardisation and documentation, the company depended heavily on key senior engineers. If they leave the company, a great deal of the knowledge leaves with them. Finally, the lack of software support also meant that the engineers spent a substantial amount of time on routine tasks, which reduces their efficiency.

Support Tool. Because of these problems with their way of working, the company wanted to develop a tool to support the design process. To promote user acceptance and ensure maintainability, the tool has to be well understood by the engineers. Over the course of the collaboration with the company, we designed and implemented a knowledge base system that provides a solution for the aforementioned challenges.

A first crucial step was to extract the engineers' domain knowledge. In this knowledge elicitation process, two parties were involved: domain experts (i.e., the company's engineers) and knowledge engineers. Both have their own area of expertise: typically knowledge engineers are not familiar with the problem domain, while domain experts are not familiar with formal knowledge representation. The proper communication between both parties is very important. We used the Decision Model and Notation (DMN) (OMG, 2019) to structure and guide the process of capturing and aligning the domain knowledge.

A second step was to use the extracted knowledge to build a system to assist both sales and engineering. However, these two departments have contradictory needs. The sales staff has limited knowledge of product design and expect the tool to propose a single complete configuration for each set of input parameters. The engineering team, on the other hand, want the system to assist them in interactively exploring the large and partially unknown design space for non-standard applications. We therefore proposed to create two independent KBs with different concerns, as can be seen in Figure 1.

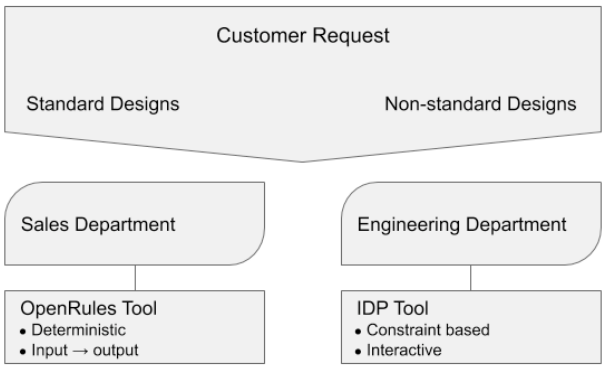


Figure 1: Two different approaches for standard and non-standard designs.

First, we made use of the DMN notation and the OpenRules[1] inference system for DMN to develop a tool for the sales staff. Second, to support the engineering staff, we developed an interactive interface to consult the constraint-based KB by means of multiple inferences like propagation, model expansion, relevance checking and explanation. This interface was implemented on top of the IDP Knowledge Base System (Bruynooghe et al., 2015). In addition to these knowledge-based functionalities, we made use of a number of machine learning techniques to allow the user to explore historically gathered data.

Knowledge in the IDP system is expressed in FO($\cdot$), an extension of First Order Logic, which is difficult to understand for domain experts that have no background in logic. To allow the engineers to understand the knowledge in the model, we developed the cDMN notation. As we will demonstrate in this paper, this notation combines the readability of the DMN standard with the expressiveness and flexibility of FO(.).

The focus of this paper is on demonstrating how the knowledge on product design and configuration can be modelled in different formal representations (DMN and FO($\cdot$)), and how this knowledge can be used to support the design process. cDMN is proposed as an extension to DMN that allows the engineers to inspect and understand the content of the knowledge base, which is crucial for the acceptance and maintainability of the tool.

This paper consolidates and expands results that have been presented at various conferences (Aerts & Vennekens, 2018, Deryck et al., 2019a). We conducted this research in collaboration with an international engineering and manufacturing company. The company wishes to stay anonymous to protect its trade secrets. It also requested that we avoid providing too much information about its products.

In the remainder of this paper, we start by discussing the technologies which we will use (Section 2), after which a running example will be introduced (Section 3). In Sections 4 and 5, we explore a decision procedure and constraint programming approach respectively. Section 6 discusses an interactive user interface to support

[1] www.openrules.com

the design process in Section 5. Then we introduce our new cDMN formalism and investigate how well this formalism is able to handle constraint knowledge in this use case in Section 7. We compare our approach to the literature and conclude in Sections 8 and 9.

2. Preliminaries

We will use two different knowledge representation frameworks: the DMN standard with its OpenRules implementation, and the IDP system with its input language FO(.). This section briefly describes both.

2.1. Decision Model and Notation

(Zhang, 2014) identified the inability to extract knowledge in a domain expert readable format as a weakness of current product configuration methods. In our project we use DMN to address this concern. This relatively new standard aims to increase the efficiency of decision-making by offering a standard notation for decision logic and "[providing] a common notation that is readily understandable by all business users[...]" (OMG, 2019, p.13). This means that the DMN notation is ideal to discuss and co-create decision logic together with domain experts.

DMN provides two interconnected viewpoints to model decisions. The Decision Requirements Diagram (DRD) is a high level model that shows an end-to-end view of a decision, from its inputs over intermediate sub-decisions to the final decision. It shows which inputs, knowledge sources and intermediate decisions are required to take the top decision. The DRD allows users to get a general overview of the domain. It can be used to delineate the borders of an automation project, and to make sure that IT and business owners are on the same page. The second viewpoint shows the underlying decision logic of each individual (sub-)decision. Although this decision logic can be modeled in multiple ways, it is typically represented in a decision table, which maps each set of inputs to the appropriate output. The individual decision tables are easily understandable with little or no prior training, which makes them suitable to discuss the decision logic between employees from different services. In fact, one of the goals of the standard is that business users should be able to maintain the decision logic without intervention of IT. As a result, business ownership of decisions increases, as well as maintainability of the application.

Figure 2 shows a typical decision table in the rows-as-rules format. The input columns have dark green headers, while the output columns at the right side of the double separation line have light blue headers.

The syntax used in the tables is called the (Simple) Friendly Enough Expression Language ((S-)FEEL), which is also part of the DMN-standard. Besides simple values, FEEL also allows numerical comparisons, ranges of values and calculations.

A decision table's *hit policy* determines how the table should be interpreted. The table in Figure 2 has the *Unique* hit policy (U), as indicated in its upper left cell. This means that each combination of input values may match at most one row. Other hit policies are *Any* (multiple rows may be applicable, but they should have the same output) and *First* (if multiple rows are applicable, only the output from the first hit is

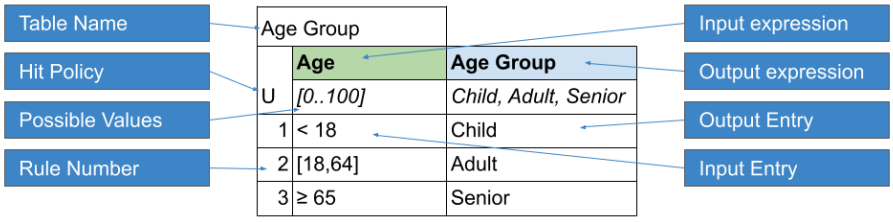


Figure 2: Components of a decision table

considered). Next to these *single hit* policies, there also exist *multiple hit* policies. In this case multiple rows can apply, and the different output values are used, e.g., to create a list.

2.2. IDP and FO(·)

IDP stands for *imperative declarative programming*, which refers to the declarative KBs on the one hand, and the ability to apply different inference algorithms to these KBs through an imperative shell on the other hand. As such, it is an implementation of the Knowledge Base Paradigm, which advocates a strict separation between declarative domain knowledge and the use of this knowledge to solve different problems in this domain. The declarative KBs are written in FO(.), a language that extends classical First Order Logic language with aggregates, types and inductive definitions (Bogaerts et al., 2018, Wittocx et al., 2008). The extensions allow to express complex knowledge, such as the fact that the size of a product equals the sum of the sizes of all components used in the product:

$$\forall p[Product] \ : \ Size(p) = sum\{c[Component] \ : \ Used(c) \ \wedge \ Size(c) = s \ : \ s\}. \quad (1)$$

The knowledge in an IDP specification is structured in three blocks, the *vocabulary*, the *structure* and the *theory*. A *vocabulary* is a set of types, functions and predicates. As usual, each type is interpreted by a set of domain elements, each n-ary predicate by a set of n-tuples of domain elements, and each n-ary function by a mapping of n-tuples to elements. In addition, FO(·) also allows three-valued interpretations of relations, in which it may be unknown whether a tuple belongs to the relation or not. A *theory* is a set of logical formulas. When all formulas in the theory T hold in a structure S, that structure is a model of the theory ($S \models T$).

With an efficient solver, a number of inference tasks can be applied to this knowledge, such as *model expansion*, *optimization*, *propagation* and so on. These inference tasks are discussed later in Section 5.5.

3. Running Example

In this section we introduce the running example, which we will use throughout this paper to demonstrate the key differences between the approaches that are presented. This

example is an extract of the complete product design case. Although it is a simplification of the real-world case, it nevertheless retains most of its essential features.

The task is to design a product that may consist of three components: a body, a spring and a spacer. The use of the spacer is optional. The engineers are given a number of specifications:

- The requested dimension of the product.
- The temperature range and pressure it should be able to handle.
- Whether there is a risk of pressure from the back side of the product (back pressure).
- The position in which the product will be installed.

They need to determine:

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

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

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

Typically, cheaper materials are weaker than more expensive ones. In general, the engineers look for the cheapest design that meets the criteria.

4. Modelling the Design Process

Each engineer has their own "standard" decision process that they follow to handle routine requests. However, this process is not standardised in the company, and engineers at different locations may do things somewhat differently. To standardize this process and to be able to automate it, the engineers' detailed technical knowledge needs to be represented in a formal and structured manner. In this section we describe the substantial knowledge elicitation effort needed to gain insight in the tacit design processes of different engineers, and to unify and formalise them in decision models.

4.1. Knowledge Extraction of the Design Process

As we described in Section 2.1, the ability of DMN to be understood by business users (in this case, the product engineers) is a key advantage. It provides a common language that is readable for both our knowledge engineers and the company's domain experts. This helps to avoid misunderstanding, ensures smooth communication, and allows certain well-delineated parts of the decision model to be assigned as "homework" to specific experts. Moreover, the product engineers' ability to not only read but also co-create a formal model, is of significant importance for the maintenance of the decision model. The engineers themselves will be responsible for this.

To create a DMN model, we organised multiple brainstorming workshops with the involved parties. Each workshop spans a couple of days and results in an initial representation of the design process for a clearly delineated application area. Participants include a number of design engineers from different locations worldwide, a 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 a brief 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 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.

For the applications that fall within the existing Visual Basic tool of the sales staff, we used a different method. In this case, we started from the existing Visual Basic code and transformed this into a DMN model. The DMN model proved to be significantly shorter (360 lines of VB code were reduced to 80 table rows spread across 20 tables), more structured, and therefore easier to maintain.

4.1.1. Results

Following the methodology outlined in Section 4.1, 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.

Overall, the DMN representation 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. Our experiences therefore confirm that DMN's readability for domain experts is a big advantage of this standard. However, there were some exceptions. Occasionally, the engineers do not follow a strict bottom-up decision procedure when making their design. For instance, 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. In Section 4.2 we discuss the work-around that we have used for this.

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

4.2. Running Example

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

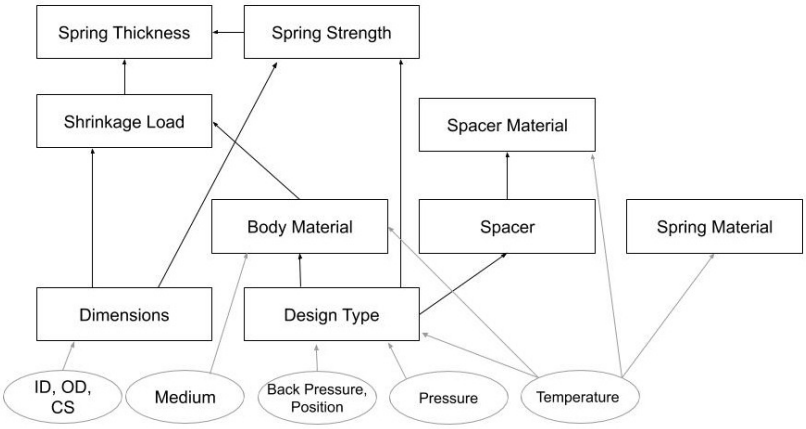


Figure 3: DRD of the product design application.

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

Figure 4: Decision table for dimensions.

Figure 3 shows the DRD of our model, which captures the general structure of the decision logic. It starts from the customer requirements (product dimensions, operating conditions) at the bottom and has the decisions that must be made at the top, with various sub-decisions in between. Each rectangle corresponds to a decision table. These can be found in Figures 4, 5, 6 and 7.

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

1. There are three relevant *dimensions*: the outer diameter (OD) of the product, its inner diameter (ID) and its cross-section (CS). The customer provides two of these, and the engineer computes the third, using the formula $CS = (OD - ID)/2$. The corresponding DMN table in Figure 4 consists of three different rows (one per dimension that might be missing), that all contain essentially the same information.
2. An initial *design type* is chosen, depending on the required operating conditions, whether there is back pressure and in which position the product will be installed. This decision fits well within the mould of a DMN decision table, as can be seen in Figure 5a. However, as we will discuss in Section 4.4, the table fails to capture the underlying reasons behind this decision.
3. A *spacer* is selected based on the design type as shown in the second decision table in Figure 5b.

Design Type					
A	**Back Pressure**	**Position**	**Pressure**	**Temperature**	**Design Type**
1	True	Pressure Accumulating	—	—	Open
2		Bi Directional	—	—	Open
3		not(Pressure Accumulating, Bi Directional)	> 150	—	Open
4			(100, 150]	≥ -50	Open
5			(100, 150]	< -50	Closed
6			≤ 100	—	Closed
7	False	—	—	—	Closed

(a) Design Type

Spacer		
U	**Design Type**	**Spacer**
1	Closed	False
2	Open	True

(b) Spacer

Figure 5: Decision tables for Design Type and Spacer.

4. Based on the temperature restrictions and design type, the best *material* is selected for the body, spring and spacer, as can be seen in Figure 6.
5. The expected *shrinkage* is computed based on the design type, dimensions, temperature and selected materials. While we omit the details, the formula for this calculation can easily be placed in the "Shrinkage Load" table in Figure 7b.
6. Based on the selected design type, the *spring strength* of a standard spring is determined. If this is enough to cope with the expected shrinkage, the standard spring is selected. Otherwise, the engineers switch to a thicker spring and recompute the spring strength of the design.
   Since DMN does not allow to recompute a value, the "Spring Strength" in Figure 7a table computes both the spring strength that the design would have with a standard spring and that it would have with a thicker spring. The "Spring Thickness" table in Figure 7c then uses these two values to decide which spring type to use. Because of this, the DRD has an edge *from* "Spring Strength" *to* "Spring Thickness". This is counter-intuitive, because in reality it is the thickness of the spring that determines the design's spring strength, not the other way around.
7. If the thicker spring from the previous step does not provide enough spring strength, the engineers' final option is to switch to another design type. Since the choice of design type is the initial choice upon which all further choices are based, this means that the engineers essentially restart the entire process from scratch.
   When no acceptable spring can be found, the DMN table "Spring Thickness" returns `null`. This alerts the users of the system that something is wrong, but the backtracking step of redoing the entire design process with a new design is not possible in DMN. Therefore, we have implemented a workaround in which the "Design Type" table is overly cautious: in low temperatures, it will always choose the more shrinkage resistant closed design. This means that the product

Body Material			
P	**Temperature**	**Design Type**	**Body Material**
	[-150.. 200]	*Closed, Open*	*M2, M1, M3*
1	[-150, -120)	—	M1
2	[-120, 80]	Closed	M1
3	[-120, 80]	Open	M2
4	(80, 100]	—	M1
5	(100,200]	—	M3

(a) Body Material

Spring Material		
U	**Temperature**	**Spring Material**
1	[-150, -50)	M1
2	[-50,150]	M5
3	(150,200]	M3

(b) Spring Material

Spacer Material			
U	**Spacer**	**Temperature**	**Spacer Material**
1	True	[-150, -50)	M1
2		[-50,150]	M5
3		(150,200]	M3
4	False	—	null

(c) Spacer Material

Figure 6: Decision tables to determine component materials.

will be over-designed in certain circumstances, but it prevents possible failure of the process.

4.3. Results

DMN is designed to be a fully executable specification and is currently supported by a number of different tools, both commercial and open source. We have created an automated design system by implementing the constructed DMN tables in the OpenRules system[2].

Our approach of basing the tool on a DMN formalisation of the engineers' design process proved successful. Not only did the engineers appreciate the intuitive nature of the DMN standard, it also made them think about how they come to a design in a given situation. Consequently, it helped them to better understand and standardize their design process. Moreover, transforming the Visual Basic tool that was originally developed for the sales staff into a DMN model, brought to light a number of errors. Without a formal representation of the knowledge, these errors would have been far more difficult to detect.

We created DMN models for six application areas. The average application contains about 30 decision tables with 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. Our OpenRules implementation generates the correct design in all of the test cases.

[2]http://openrules.com

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

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

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

(b) Calculation of Shrinkage Load

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

(c) Decision table for the Spring Thickness

Figure 7: Decision tables to check whether a design can cope with the expected shrinkage.

Computing a design takes about 0.3 seconds single core on an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.

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

4.4. Discussion

According to Shafiee et al. (Shafiee et al., 2020), configuration projects differ from other IT projects in several ways. It is more difficult to delineate the scope of a product configurator because of the inherent complexity and diversity of the relevant knowledge, and because of frequent changes to it. Moreover, because such applications require a large amount of detailed information, projects require heavy involvement of product engineers and extensive documentation that must be understandable for both IT and business users. As described above, the use of DMN facilitated the delineation of the scope, facilitated the involvement of the engineers, and resulted in an extensive knowledge base of the available information. However, both literature and our case also point towards some downsides to the use of DMN.

First, as mentioned in Section 4.2, a few aspects of the design process do not fit readily into the DMN model. This lack of expressivity has been recognized earlier, e.g., by Car (Car, 2018), who points out that the lack of expressivity does not hamper automated decision making as such, but does force the modelers 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.

Second, DMN consists of a strict hierarchy of mandatory decisions. It cannot represent interwoven, optional or cyclical decisions. In our case, the selection of the design type and spacer is an example of two decision tables that are interwoven. While some circumstances require a closed design, other circumstances require the use of an open design with a spacer. Sometimes both designs are possible. The two decisions involved are selection of the design (open or closed), and selection of the appropriate spacer. Both decisions are inherently connected, but the DMN standard forces to impose some arbitrary order on them. Our case study also contains decisions that are optional, in the sense that, in certain circumstances, the decision need not to be made. For instance, if the design does not contain a spacer, there is no need to decide on the spacer material. There is currently no way to model this in DMN, although this has been identified as one of the shortcomings of the standard (Mertens et al., 2015).

Third, DMN is not able to represent background information or constraints. This typically forces one to mix different kinds of knowledge within a single table. In our application, physical constraints and preferences of the company are mixed in one table, as for instance in the decision table "Design Type" in Figure 5a. This decision table could be explained in any of the following ways:

- A closed design is always preferred, but it is not usable in rules 1, 2, 3 and 4.
- An open design is always preferred, but it can only be used in rules 1, 2, 3 and 4.
- An open design is preferred if there is back pressure, while a closed design is preferred in all situations when there is no back pressure.

Mixing constraints and preferences in this way reduces the maintainability of the tables. 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, without knowing the underlying rationale of this table, 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 (Calvanese et al., 2019), or a constraint addition in OpenRules (Feldman, 2011).

Fourth, the available inference engines for DMN support only a single inference task, namely that of computing the "output" decision variables given all the input variables. This goes further than a mere lack of vendor interest in developing tools capable of other inference tasks. Car (Car, 2018) considers this a shortcoming of the standard itself, because it does not use more advanced modelling standards that would allow other forms of inference. Either way, the fact remains that the current tools offer limited functionality. For example, one disadvantage of DMN that is often cited is its inability (in combination with BPMN) to model a series of questions (Sooter et al., 2019). This is a common problem, that also appeared in one of our earlier cases and that we then resolved outside the DMN tool (Deryck et al., 2019b). In Section 6, we discuss the

different forms of inference that are available in the tool that we have developed to interactively support the engineers.

Fifth, a DMN model decomposes a final decision in hierarchical structured subdecisions. This reduces the complexity of the decision, and leads to smaller and easier to read decision tables. A downside of this approach is that it is not possible to talk about global properties of the design. E.g., the cost of a component depends on its material. The cost of the design depends on the cost of the components that are used. However, for some components the use of a better material might eliminate the need for a particular additional component. This interdependency means that we cannot in general find the cheapest global design by making a sequential series of local decisions.[3]

Sixth, the entire DMN approach of course assumes that there *is* a decision procedure to model. The approach cannot handle uncertainty and is limited to pre-defined decisions with known in- and outputs (Biard et al., 2015). For the design of products in new and challenging application areas, the design process is unknown. This means that DMN cannot be used at all, even though (some of) the information from known areas may still be applicable.

Finally, another weakness of the DMN standard identified in recent literature is the limited legibility of large tables, thus hampering manual validation (Mertens et al., 2015). Alternative, multiple tools for automatic verification exist (e.g.; (Hasić et al., 2020)). Our solution does not focus on this problem, as we did not observe this problem in our case. All large tables in our applications are converted data tables that contain no decision logic, so there is no need for users to be able to interpret the tables.

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

5. Modelling the Underlying Constraints

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

In general, the design process followed by the engineers is governed by a number of physical constraints (e.g., a material $M1$ can only be used in temperatures $\leq 100°C$) and preferences (e.g., material $M2$ is preferred over material $M1$, perhaps because it is cheaper or more durable). In order to develop a decision support system that can also provide useful information for challenging new application areas, we need to make direct use of these underlying constraints and preferences, rather than of the engineers' existing design process. These contain more information than is explicitly present in the design procedure, because they also explain *why* certain designs are impossible. As such, it is not possible to automatically deduce these constraints from the design

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

procedure. Instead, coming up with them requires additional discussions with the design engineers.

5.1. Knowledge Extraction of the 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. The elicitation of this information proved difficult. The engineers often did not know where to start and discussions tended to be chaotic. To remedy this, we used the DMN models to structure the process.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.

This use of the DMN tables provides a structured way of working, in which different topics are addressed in a meaningful order and we can be sure that no relevant constraints 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.

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 5a suggests that the closed design is the preferred choice, with the other rows describing circumstances in which this preferred choice is not possible. Additionally, considerations that were mentioned during the workshops to construct the DMN models may provide further clues. In practice, we have found that we can construct most of the constraints without help of the engineers and only need them to verify and 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.

$$DesignType = \text{"}Open\text{"} \Leftrightarrow ReleasePressure. \tag{2}$$

$$Location = \text{"}PressureAccumulating\text{"} \Rightarrow ReleasePressure. \tag{3}$$

Both kinds 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.

Even though most of the DMN decision tables can be interpreted individually, in some cases, multiple decision tables are interwoven. As a result, the same constraints underlie several of these decision tables at once. Section 5.3 handles a detailed example of this.

5.2. Knowledge Extraction of the 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. Our optimization framework uses a three-level structure to this end. The highest level consists of a lexicographic ordering of the solution space, according to a sequence of criteria $m_1, ..., m_n$. 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:

$$lexico\big(m_1, ..., m_n, pareto(m'_1, ..., m'_k), max(\sum_i \alpha_i m''_i)\big) \tag{4}$$

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

5.3. Running Example

In this section we apply the constraints and preferences approach to the running example described in Section 3. We use the FO($\cdot$) language to represent them. We explain how the formulated constraints relate to the DMN model that we presented in Section 4.2.

Dimensions. A constraint such as $CS = (OD - ID)/2$ expresses a relation between three variables. In DMN, we had to designate some of these variables as input and the others as output, leading to the three separate table rows in Figure 4. Here, we can just use the single constraint to derive the value of the remaining variable from that of the other two.

Design Type and Spacer. The restrictions on *Design Type* and whether a *spacer* is used are interwoven and more complex. Discussions with the engineers have revealed that the decision tables in Figure 5 can be explained as a combination of the following constraints and preferences.

Constraints:

- Only open designs are able to release pressure:

$$DesignType = \text{"}Open\text{"} \Leftrightarrow ReleasePressure. \tag{5}$$

- When the product is placed in a pressure accumulating location, it should be able to release pressure:

$$Location = \text{"}PressureAccumulating\text{"} \Rightarrow ReleasePressure. \tag{6}$$

- It is impossible to use a spacer in combination with a closed design:

$$\neg\big(DesignType = \text{"}Closed\text{"} \wedge ComponentUsed(Spacer)\big). \tag{7}$$

- A spacer is necessary if the back pressure is higher than 150 bar:

$$BackPressure \wedge Pressure > 150 \Rightarrow ComponentUsed(Spacer). \tag{8}$$

- In the "Bi Directional" location, the component tends to move back and forth excessively, so in order to avoid damage, a spacer is always necessary:

$$Location = \text{"}Bi\text{-}directional\text{"} \Rightarrow ComponentUsed(Spacer). \tag{9}$$

- Each design should always have a body and a spring:

$$ComponentUsed(Body) \wedge ComponentUsed(Spring). \tag{10}$$

Preferences:

1. If the back pressure is between 100 and 150 bars, it is recommended but not mandatory to use a design with a spacer.

2. Closed designs tend to be cheaper and outperform open designs, so they are the preferred type of design.

The first line in table "Design Type" in Figure 5a is a result of combining constraints 5 and 6. The component should be able to release pressure and since closed designs cannot do that, an open design is the only option.

The second row is a combination of constraint 7 and constraint 9. In the "Bi Directional" location a spacer is always necessary, and since it is impossible to have a spacer in closed designs, the only remaining possibility is an open design.

Analogously, the third line in the decision procedure can be obtained by combining constraint 8 and 7.

The combination of constraint 7 and preference 1 would suggest selecting an open design if the back pressure is between 100 and 150 bars. However, in the decision table, a distinction is made between row 4 and 5 based on the temperature. This is the previously mentioned work-around to avoid running into cases in which the shrinkage cannot be addressed by simply adding a thicker spring to an open design. By contrast, our constraint representation avoids the need for such a work-around (see below).

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

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

$$\forall c[Component] : ComponentUsed(c) \Leftrightarrow Material(c) \neq null. \tag{11}$$

$$\forall c[Component] : ComponentUsed(c) \Rightarrow \\ MinTemp(Material(c)) \leq Temperature \leq MaxTemp(Material(c)). \tag{12}$$

The specific values for $MaxTemp$ and $MinTemp$ of each material are predefined, and can be found in Figure 16b.

Not every material can be used for each component. This is expressed in the predefined *PossibleMaterial* predicate:

$$\forall c[Component] : ComponentUsed(c) \Rightarrow PossibleMaterial(c, Material(c)). \tag{13}$$

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

$$DesignType = Closed \Rightarrow Material(Body) \neq M2. \tag{14}$$

These constraints still leave considerable freedom in selecting a material. In our original DMN model (Figure 6), we selected a specific material for each set of circumstances, based on the two physical constraints, and on the fact that the engineers prefer to use cheaper materials when possible. With constraint-based reasoning, we can separate these two concerns, expressing the constraints in the tables mentioned above, and subsequently expressing an optimization criterion as discussed in Section 5.3.1.

To illustrate how these constraints and preferences correspond to the "Body Material" table in Figure 6a, we discuss one row in this decision table. Based on the constraints and preferences, row 5 can be explained as follows: in this temperature range, all three materials M1, M2 and M3 are possible; however, if a closed design is selected, material M2 can no longer be used. From the remaining materials (M1 and M3), the cheapest (M1) is then selected. This is indeed the outcome of the decision table. All other rows in the table can be explained analogously.

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

$$ShrinkageLoad \leq SpringStrength. \tag{15}$$

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

5.3.1. Preferences

In the current version of the application, the optimization framework described in Section 5.2 is implemented as a weighted cost function. Lexicographical criteria are modelled as n-category integers $[1,n]$, and the corresponding weight is set to a sufficiently high value to ensure that no combination of other criteria can compensate for a non-optimal value. Another concern is to ensure that only pareto optimal solutions are proposed, even for categorical criteria. This is implemented by introducing ascending numerical values for those categorical criteria. By ensuring all pareto criteria are included in the weighted sum, only pareto optimal solutions will be proposed.

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

- The *cost* of a design can be determined by adding up the used material cost of each component, listed in Figure 21. Note that a design without a spacer will be cheaper, since it requires less material. Moreover, thicker springs are harder to fabricate, which makes the production cost higher.

- The *durability* of a design depends on a number of factors. When a product without spacer is used in a back pressure environment, it will wear faster. When this pressure is higher than 100 bar, the product is damaged almost immediately. Moreover, a product wears because of the spring strength.

- The *leak tightness* of a product is proportional to the spring strength. For example, closed designs typically have a higher leak tightness than open designs.

- *Availability*: certain materials could be out of stock and should be avoided if possible. This is the most important criterion in the lexicographical ordering.

In the framework proposed in Section 5.2, these criteria are combined as

$$Lexico(Availability, Pareto(Durability, LeakTightness, Cost), Min(Cost)). \quad (16)$$

As explained, our current tool implements this by a weighted sum function. The weights can be tweaked to favour one criterion over the others.

The following extract shows the logic to construct the optimisation term. Here, we use the FO($\cdot$) notation of $sum\{\bar{x} : \varphi(\bar{x}) : F(\bar{x})\}$ to denote $\sum F(\bar{x})$ over all $\bar{x}$ such that $\varphi(\bar{x})$.

$$Availability = sum\{c : IsUsed(c) \wedge \neg LocallyAvailable(Material(c)) : -100\} \quad (17)$$

$$\begin{aligned} Durability = \\ sum\{\ &: \neg IsUsed(Spacer) \wedge BackPressure \wedge Pressure < 100 : -5\} + \\ sum\{\ &: \neg IsUsed(Spacer) \wedge BackPressure \wedge Pressure \geq 100 : -100\} + \\ &-1/5 \times SpringStrength. \end{aligned} \quad (18)$$

$$LeakTightness = 1/2 \times SpringStrength. \quad (19)$$

$$\begin{aligned} Cost = \\ &sum\{c : IsUsed(c) : Cost(Material(c))\} + \\ &sum\{\ : SpringThickness = Thick : 5\} + \\ &sum\{\ : SpringThickness = Standard : 2\}. \end{aligned} \quad (20)$$

$$OptimisationFunction = Durability + LeakTightness - 5 \times Cost + Availability. \quad (21)$$

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

The combination of constraints and preferences defined in this section produce the same outcome as the DMN procedure of Section 4.2. For example, an open design will only be used when a spacer is not necessary (because it is less leak-tight), cheaper

materials are selected when they can handle the operating conditions, and so on. A more detailed comparison is made in Section 7.7.5.

Separating the preferences and constraints increases the flexibility in a number of ways.

- The optimisation criteria can be easily adjusted by tweaking the weights, depending on the customer's priorities.
- When a material's cost or local availability changes, this can easily be updated in the interpretation of the function cost or of the predicate *Locally Available*. There is no need to change the constraints or preferences themselves.
- Additional criteria can be added in a modular way.

5.4. Missing Data

In a different application area, we encountered an additional challenge. Certain data was unknown, e.g., the property $LifeExpectancy$ is known for some designs, but not for others. The company's engineers indicated that this is indeed the current situation. It would take great effort to determine these missing values and complete the data in the property tables. Therefore, a solution should be developed that allows our system to handle the missing data.

We introduce a small example to illustrate the mechanism we propose to cope with missing data. Each of the properties that has missing values becomes a partial instead of a total function, whose value is only defined when the property is known, e.g.,

$$LifeExpectancy = \{D1 \rightarrow x; D3 \rightarrow z\} \tag{22}$$

when the $LifeExpectancy$ of design $D2$ is unknown. Then, the constraint states that $LifeExpectancy(UsedDesign) > RequestedLifeExpectancy$ is changed to one of the following logical formulas:

$$\forall x : LifeExpectancy(UsedDesign) = x \Rightarrow x \geq RequestedLifeExpectancy. \tag{23}$$

$$\exists x : LifeExpectancy(UsedDesign) = x \wedge x \geq RequestedLifeExpectancy. \tag{24}$$

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

The engineers could not specify a general desired behaviour of the system, since this greatly depends on the situation. They would like to be able to switch between those two behaviours manually. Therefore, we slightly adapted our approach to give them this choice. We introduced a new predicate in the vocabulary: $AllowUnknown(Property)$ and changed the constraint to:

$$\begin{aligned} &\big(\exists x : LifeExpectancy(UsedDesign) = x \wedge x \geq RequestedLifeExpectancy\big) \vee \\ &\big(\neg\exists x : LifeExpectancy(UsedDesign) = x \wedge AllowUnknown(LifeExpectancy)\big). \end{aligned} \tag{25}$$

If $\neg AllowUnknown(LifeExpectancy)$, designs with an unknown $LifeExpectancy$ are not considered and vice versa.

5.5. Implementation of a Constraint-based Approach

We used the IDP knowledge base system (Bruynooghe et al., 2015) to implement the constraint-based design system. It offers different algorithms, implementing a number of logical inference tasks, based on Answer Set Programming (ASP), Logic Programming (LP) and SAT solving technology. In recent editions of the ASP Competition (Alviano et al., 2013), IDP proves to be competitive with other state-of-the-art ASP systems, though typically somewhat slower than systems such as Clasp.

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

Second, as IDP is an implementation of the knowledge base paradigm, it provides support for different logical inference tasks on the same knowledge base. This can be used to provide a multitude of functionalities to the end user. Such flexibility is not offered by constraint-programming languages such as MiniZinc (Nethercote et al., 2007).

We can reproduce the functionality of the OpenRules implementation using our constraint-based representation in IDP, by using the inference tasks of *Model Expansion* and *optimisation*. Other inference tasks are used in our GUI which we discuss in Section 6.

Model Expansion. To compute a design, we apply the logical inference task of Model Expansion (Mitchell et al., 2006). This takes as input a theory T and a structure S_{in} for part of the vocabulary of T, and the goal is to produce a structure S_{out} for the remaining part of the vocabulary such that $S_{in} \cup S_{out} \models T$. In our case, the structure S_{in} describes the problem specification, by providing an interpretation for constants such as $Temperature$, $Pressure$ and $Location$; the structure S_{out} then describes a design, by providing an interpretation for constants such as $DesignType$ and functions such as $Material$.

Optimisation. IDP's optimisation functionality allows us to specify a numerical term t together with a model expansion problem (T, S_{in}). IDP will then compute not just any solution to the model expansion problem, but the solution S_{out} that, in addition to being such that $S_{in} \cup S_{out} \models T$, also minimizes the value $t^{S_{in} \cup S_{out}}$ of this term. In our case, the term is the weighted sum $Optimisationfunction$, as defined in Section 5.3.1. IDP implements this inference task with an optimisation loop, which iteratively produces better solutions by each time adding a new constraint that the next solution must have a lower score than the previous solution. This method is typically also used in, e.g., ASP solvers.

5.6. Results

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

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

Second, to verify that the constraints are not too weak, we also check that the design $D(I)$ proposed by the DMN model D is among the optimal solutions of this model expansion problem, i.e., that $t^{S_I \cup S_{D(I)}} \leq t^{S_I \cup S'}$ for any other solution S' to the model expansion problem (T, S_I), where t is the optimisation term that is minimised. This checks that 1) the constraints do not fail to rule out designs with a higher score that are in fact impossible and 2) that the weights used in the optimisation criterion are assigned correctly.

We implemented both of these checks using IDP. We first transformed the DMN model to IDP syntax as described in (Dasseville et al., 2016). We then used IDP to perform the required checks on the relation between the IDP theory derived directly from the DMN model and the IDP theory that represents the constraints.

The first check initially revealed a small number of errors in the constraint-based representation. After minor fixes, it was concluded successfully. In the second check, IDP proposed a different design than the DMN model for a number of cases. A thorough analysis of these cases revealed that the outcome of the constraint-based model was correct, and that the DMN solution was sub-optimal. This non-optimality turned out to be caused by the obligation of making decisions in a fixed order. When using the constraint-based method, on the other hand, the solver is free to make decisions in any order, allowing it to find a better scoring global optimum.

For an application area with about 30 variables the IDP system typically finds the optimal design in about 3.15 seconds on one core of an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.

5.7. Discussion

The constraint representation has a number of interesting advantages. By modelling the required characteristics of the product as constraints rather than specific rules, the outcome is usually a set of solutions rather than a single solution. Hence, the constraint approach enables flexible 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. It proves to be complicated to write down individual constraints in the correct syntax. While we have used classical FO because we believe it is quite understandable for untrained domain experts, it is still more complex than the simple table-based DMN format. Apart from this, it is inherently more complex to reason 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. For example in Section 5.3, the design type is influenced by a multitude of constraints. Finding out which constraints influence a particular aspect of the design and determining their joint outcome is not a straightforward task and we find this often confuses the domain experts.

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

6. Interactive Decision Support

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

We reuse and extend the IDP AutoConfig tool (Dasseville et al., 2016), which aims to provide a generic knowledge-based configuration tool that can easily be applied to different domains. It supports the principles of the knowledge base paradigm by allowing multiple inference tasks to be applied on the same knowledge base. This notably allows the users to perform basic tasks, but also helps them to answer more advanced questions. The tool's main screen displays a three-valued structure which represents the current state of the configuration process. In such a three-valued structure, each ground atomic fact $P(d_1,...,d_n)$ (with P an n-ary predicate symbol and the d_i domain elements of the appropriate type) or $F(d_1,...,d_n) = d_{n+1}$ (with F an n-ary function symbol) can be either certainly true (ct), certainly false (cf) or unknown (u). In the initial state of the system, all atoms that describe that design are unknown. While the system is being used, increasingly more atoms are assigned a ct or cf value, either by the user or by the system itself, in response to user actions. This narrows down the final design.

The tool does not force the engineers to enter information in a fixed order. Instead, it allows them to specify any piece of information at any stage in the configuration process. At each point in time, the tool computes all consequences of the information

it has received so far. This way of working stands in sharp contrast to how the DMN system can be used.

Because IDP lacks support for non-linear calculations with real numbers, the calculation of the shrinkage is currently omitted from the knowledge base used in this tool. Moreover, even calculations which are supported may significantly slow down the system. If we include our optimisation criterion (see Section 5.3.1) in the theory, the required calculation time for a single propagation step grows to over 20 seconds. For this reason, our current prototype also excludes this criterion. Both limitations originate from the implementation technology used in IDP3 and are not inherent to our approach. Further development to the IDP solver will solve these issues.

6.1. Existing Functionalities

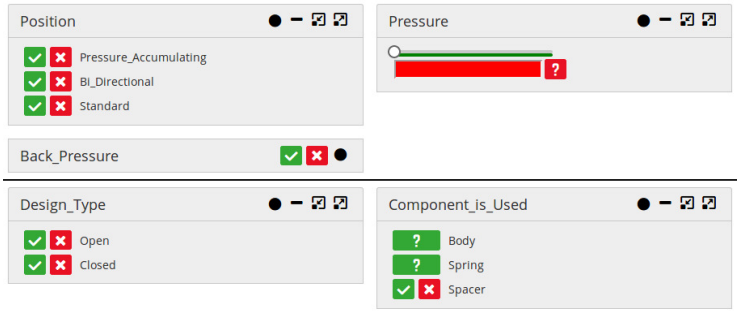
In this section we discuss how we applied the functionalities that were already available in the AutoConfig tool to our case study.

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

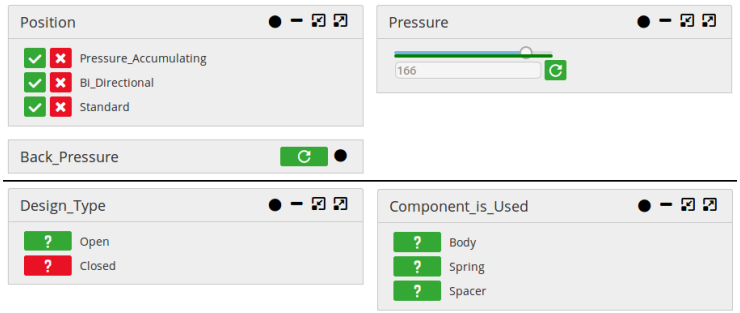
This functionality is perceived to be very useful in this specific use case because it allows the system to guide the user in exploring the search space. The system makes a distinction between the user's selection and propagated values. The user can *undo* their own selections, but not the propagations made by the system (because they are, by definition, forced by the selections that were made). All the user can do with propagated values is ask for an explanation of why this value must be assigned (see below). For this reason, user selections of propagated values are indicated with Undo and Question-mark symbols, respectively, as can be seen in Figure 8. The reason for this distinction is that they can ask for an explanation of the propagated values.

Explanation. For each of the propagated atoms it is possible to request an explanation of why this atom has been propagated as certainly true or certainly false. This explanation consists of the user-selected atoms that the system used to deduce this truth value. Typically, the user inputs environmental variables under which a design should be able to operate. As a result, some designs can no longer be used.With the explanation functionality, IDP is able to identify which (combination of) input variables causes that design to be infeasible. An example of this can be seen in Figure 9. This functionality is particularly important for the user acceptance of the proposed product design.

Model expansion. The *model expansion* functionality allows the user to ask the system to complete the current three-valued interpretation into a two-valued model M of the knowledge base T, by assigning truth values ct or cf to the unknown atoms such that ($M \models T$). In principle multiple models are possible, and the user has no control over which one is chosen (see Section 5.5). In the specific context of our use case, this inference was not found to be very useful.



(a) The initial state



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

Figure 8: Propagation in the AutoConfig tool.

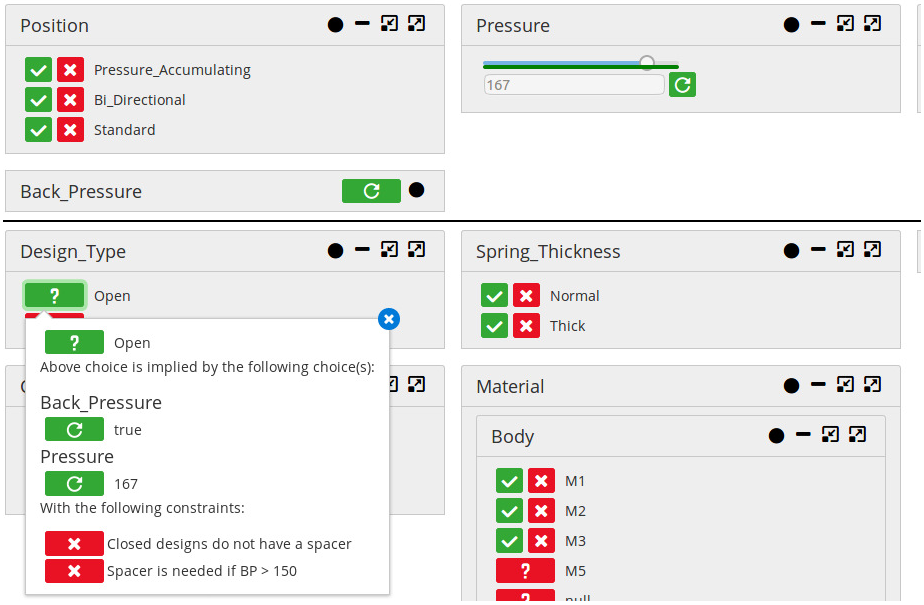


Figure 9: Explanation in the AutoConfig tool, consisting of the user-selected atoms (top) and constraints (bottom).

Optimization. The AutoConfig tool offers the possibility to optimize a predefined term in the knowledge base, as described in Section 5.5. This allows the system to compute an optimal model. This feature is useful in the context of product design to determine an optimal design for a specific set of requirements. By introducing multiple terms, it is possible to optimize according to different criteria, such as the lowest production cost, best durability or best overall design.

Relevance. The AutoConfig tool offers the ability to specify a *core* of the knowledge base. Such a core is a set of atoms such that the goal of the configuration process is to determine the truth value of these atoms. Once this has been done, the truth value of the remaining atoms is irrelevant. The relevance functionality allows the AutoConfig tool to determine which predicates and functions may still influence the value of the problem's core, and grey out the irrelevant symbols. This functionality was found useful because it prevents the engineers from having to make choices that are no longer relevant to complete the design.

6.2. Additional Functionalities

Even though the AutoConfig interface 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 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 10 illustrates our *comparison* method. It shows a comparison made between an open and a closed design. Our method works by first using *propagation* to derive the consequences of each of the possible choices. Then the resulting three-valued structures are compared and all differences are displayed. 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.

Extended Explanation. The AutoConfig tool 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

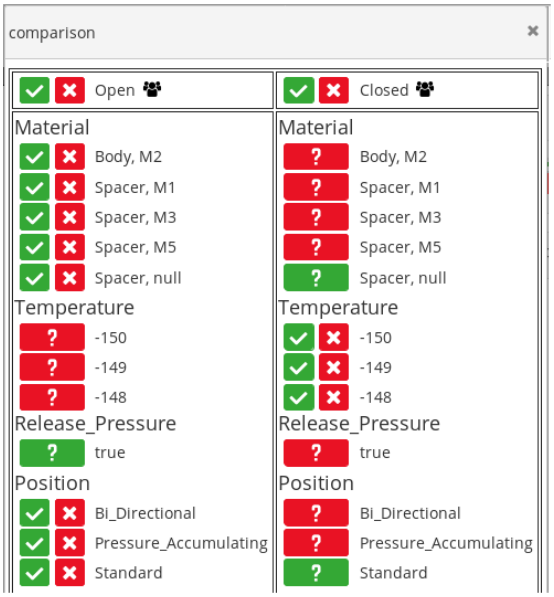


Figure 10: Comparison between an open and a closed design in the AutoConfig tool.

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. As shown in Figure 9, a complete explanation thus consists of a set of user-choices on the one hand, and a set of annotations of constraints on the other hand.

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*. 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, as shown in Figure 9. Once a constraint has been deactivated, it is important to visually indicate this, as can be seen in the upper right corner of Figure 11, so the engineer does not forget about it and possibly creates an erroneous design.

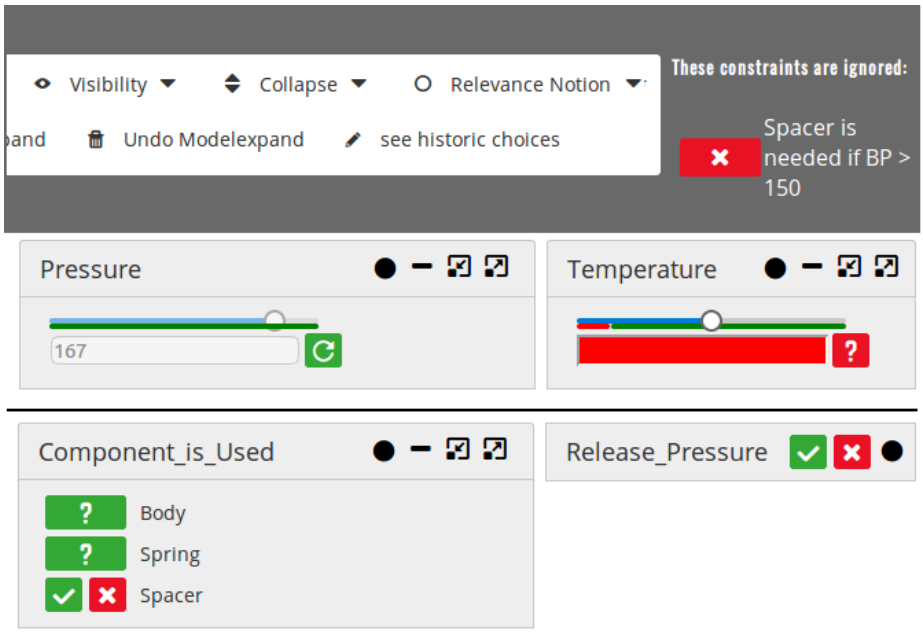


Figure 11: Visual representation of when a constraint is deactivated in the AutoConfig tool (detail of upper right corner).

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 12.

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.

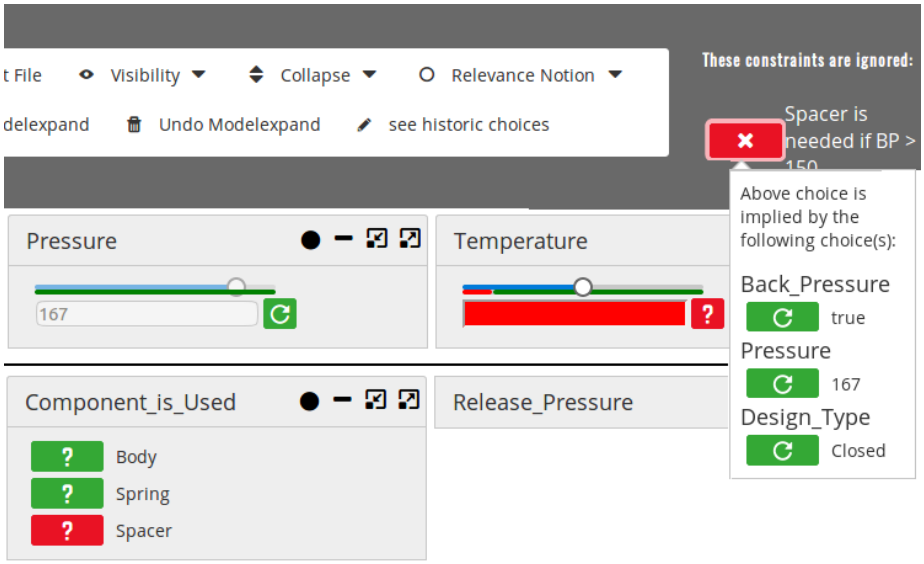


Figure 12: Indication of a constraint that cannot be reactivated in the AutoConfig tool (detail of upper right corner)

6.3. 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 (Van Daele et al., 2019). 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 13. Additionally, a summary of all applicable solutions can be found in the right-most column.

6.4. Evaluation

6.4.1. 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 bits and pieces of the interface and its functionalities. After incorporating their suggestions in the AutoConfig tool, we organised an evaluation workshop to find out if the project goals were met. Five

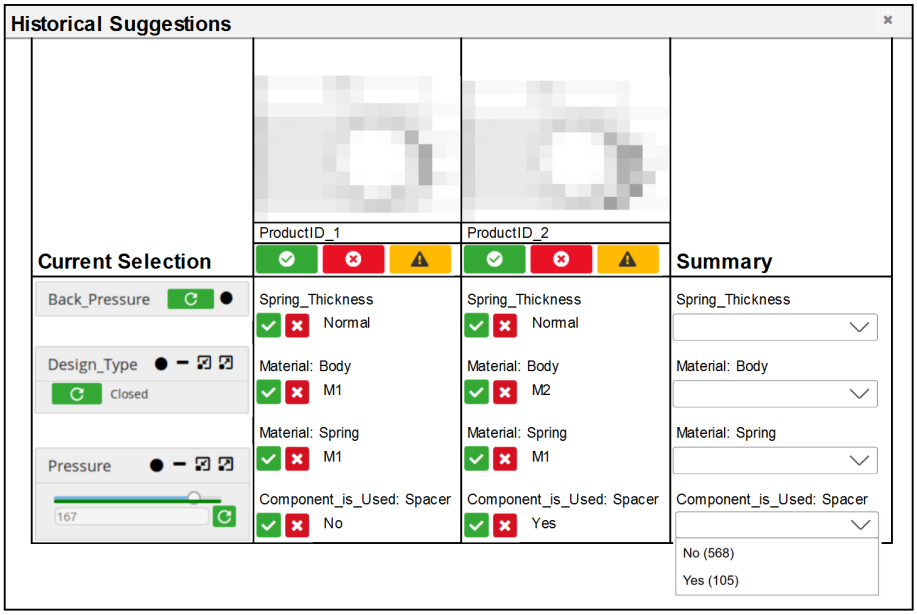


Figure 13: Browsing historical products in the AutoConfig tool (truncated image).

engineers with different levels of exposure to the tool and experience in the tested application area, were invited to use the tool. 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 evaluation consists of three parts: the expected usage of the tool, its benefits, and its potential users. The results are described below.

6.4.2. 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 ensuring 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.

6.4.3. 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 does not translate into a quantifiable time gain. For example, even though the tool helps to consult historical designs, this remains a time consuming task. The reason is that the historical database contains a large number of products. While the 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.

6.4.4. 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.

6.4.5. 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.

7. cDMN: A user-friendly representation of constraint knowledge

The FO($\cdot$) representation with constraints 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. The largest challenge for the constraint-based approach, is to extract the actual domain knowledge from domain experts. Not only is it often hard to formulate tacit knowledge, we currently also lack a formal language that allows domain experts to understand the knowledge captured in the constraint model. This opens the way for misunderstandings and prevents the domain experts from working on the model without the presence of a knowledge engineer. This stands in stark contrast to our knowledge elicitation methodology for the decision process for the well-known applications, in which the DMN standard helped us to avoid precisely these problems.

In this section, we propose an extension of the DMN standard, such that the notation becomes better suited for complex real-life situations. We do this by allowing FO constraints to be represented in a simple and readable manner. We call this extension cDMN. The full specification and examples can be found in the gitlab repository (Vandevelde & Aerts, 2020). The development of cDMN was previously discussed in a conference paper (Aerts et al., 2020).

This section first presents the building blocks of the cDMN notation. We then introduce its formal semantics. Finally, we demonstrate the use of cDMN in our running example.

7.1. Constraint Tables

DMN is well suited for applications with a clear hierarchical decision procedure to uniquely determine the value of some output. It imposes restrictions on the construction of the decision model to ensure that a unique value is always assigned to each output of each table. 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 is assigned to the output. In addition, a default value can be assigned to an output column, which will be the value of this output if none of the rows match the input conditions. If no rows match and there is no default specified, the output is assigned the special value *null* to indicate an error in the specification.

cDMN extends the scope of DMN to constraint reasoning. Instead of defining a single solution, we enumerate the conditions that possible solutions must satisfy. Hence, the outcome of the constraint-based model is not a single, deterministically defined solution, but a solution space with various possibilities. From this space, any solution or the most optimal solutions with respect to some criterion, may be selected.

To represent constraints in a tabular format, we introduce constraint tables in cDMN. The structure of such a table follows that of regular decision tables with input and output columns, and constraints as rows. Each row in a constraint table represents a logical *implication*. If the input conditions are satisfied, then the output conditions must also be satisfied. Conversely, if none of the input rows are applicable, the output can take on any value, as opposed to being forced to *null* as in regular decision tables.

In constraint tables, output columns can contain any S-Feel statement, whereas in decision tables, only single atomic values (such as integers, floats and strings) can be used. This greatly improves expressivity.

Constraint tables can be recognized by their hit policy, which is the *Every* hit policy, denoted as *E**. It expresses that for every row in the table, the implication must be satisfied.

The following example illustrates the difference between a decision and constraint table. As shown in (Calvanese et al., 2016), the first line of the decision table in Figure 14b is logically equivalent to:

$$\forall x : SpacerIsUsed = x \Leftrightarrow Design = Closed \wedge x = False \vee Design \neq Closed \wedge x = True. \tag{26}$$

In other words, the only way to obtain $Spacer = False$ is by having the input condition $Design = Closed$. We compare this to the interpretation of a constraint table. The first row in the "Spacer is Used" constraint table in Figure 14a is interpreted as the logical implication $Design = Closed \Rightarrow SpacerIsUsed = False$. In other words, when $Design \neq Closed$, a spacer may or may not be used.

7.2. Expressive Data Types and Glossary

In logical terms, the input and output variables used in DMN are all constants. In order to allow complex logical formulas to be expressed, cDMN allows the user to define a complete FO vocabulary with types, functions, relations, booleans and constants in the *glossary*. This glossary consists of one table per kind of symbol. The glossary for our running example can be found in Figure 15.

Spacer is Used					
E*	**Design Type**	**Position**	**Back Pressure**	**Pressure**	**Spacer is Used**
1	Closed	-	-	-	False
2	-	Bi Directional	-	-	True
3	-	-	True	> 150	True

(a) Constraint Table for the use of a spacer

Spacer Is Used		
U	**Design Type**	**Spacer Is Used**
1	Closed	False
2	Open	True

(b) Decision Table for the use of a spacer

Figure 14: Constraint and Decision Table for Spacer

- The *Type* table defines all data types used in the model. Each type is defined by its name, its base type (string, int or float) and the different values that are possible for this type. E.g., the first line of the types sub-glossary in Figure 15 defines a type called Component, which can take the values `Body`, `Spacer` or `Spring`.
- The *Functions* table contains the function symbols of the vocabulary. Such a function maps a number of arguments onto a value. Functions are declared using the notation `FunctionName of ArgumentType and ArgumentType`. The range of the function is specified in the Type column. For example, the first line of the Functions table in Figure 15 introduces a function called `Material` which maps one argument of the type `Component` to a `Material`.
- In the *Relations* table, relations on one or more types are defined, e.g., `Component is Used` denotes whether a component is used or not.
- Constants (i.e., zero-arity functions) can be defined in the *Constants* table, e.g., we introduce a `CS` of the type `Dimension`.
- Boolean variables are introduced in the *Booleans* table, e.g., `Release Pressure` denotes whether the product is able to release pressure.

7.3. Header expressions and quantification

Not only does cDMN allow more complex data types, it also allows more complex expressions to be put in the column headers of decision and constraint tables. Besides constants, the following expressions are allowed as headers:

- a type *Type*;
- an expression of the form "*Type* called *name*";
- an expression of the form "*Function* of arg_1 and ... and arg_n", where each of the arg_i is another header expression;

Types		
Name	**Type**	**Values**
Component	string	Body, Spring, Spacer
Material	string	M1, M2, M3, M5, M4
temperature	int	[-200,200]
Dimension	float	[0, 10]
...	...	...

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

Booleans
Name
Release Pressure
Back Pressure

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

Relations
Name
Component is Used

Figure 15: Extract of the cDMN glossary for the product design application.

- an arithmetic combination of header expressions (such as a sum).

The *Type* and *Type called name* expressions are called variable introducing headers. They are used to introduce universally quantified variables. With the use of quantification, compact tables can be created, in which one row is applicable for all the elements of a type. If the same type or variable name reappears further in the table, it refers back to this universally quantified variable. An example of this can be seen in Figure 16a, where a variable of type `Component` is introduced and reused throughout the table. It states that each component, if it is used, should be able to handle the operating temperature (i.e., the operating temperature should fall between the minimal and maximal temperature of the material used for that component).

7.4. Data table

Another novelty introduced by cDMN are data tables. The motivation for this kind of table comes from the fact that usually two different types of knowledge are relevant for a decision problem. The first is the decision logic or decision procedure. This knowledge can be specified in the decision tables and constraint tables. A second type of knowledge is raw data, e.g., material properties such as cost and temperature ranges. It is possible to specify this type of knowledge in decision tables. However, using data tables has the advantage that decision logic is separated from the specific problem instance at hand (e.g., the particular materials that are considered). Figure 16b shows an example of such a data table in the context of our application.

7.5. Formal semantics

In this section we describe the formal semantics of cDMN by translating it to the FO(·) language used by the IDP system.

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

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

(a) Temperature restrictions on the selection of materials.

Data Table: Temperatures			
	Material	**MinTemp of Material**	**MaxTemp of Material**
1	M1	-150	100
2	M2	-120	80
3	M3	-50	200
4	M5	-50	150

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

Figure 16: Tables for the selection of component materials.

Decision tables retain their usual semantics as described by Calvanese (Calvanese et al., 2019). We briefly recall these semantics. Each cell of a decision table (i, j) corresponds to a formula $F_{ij}(x)$ in one free variable x. For instance, a cell "≤ 50" corresponds to the formula "$x \leq 50$". A decision table with rows R, input columns I and output columns O is a conjunction of material implications:

$$\bigwedge_{i \in R} \left(\bigwedge_{j \in I} F_{ij}(H_j) \Rightarrow \bigwedge_{k \in O} F_{ik}(H_k) \right) \bigwedge \left(\neg \bigvee_{i \in R} \bigwedge_{j \in I} F_{ij}(H_j) \Rightarrow \bigwedge_{k \in O} H_k = null \right) \tag{27}$$

where H_j is the header of column j. Each conjunct corresponds to one row, except for the last one, which forces the output to *Null* if no rows are applicable. For example, the table in Figure 14b corresponds to the logical formula $(DesignType = Closed \Rightarrow SpacerIsUsed = False) \wedge (DesignType = Open \Rightarrow SpacerIsUsed = True) \wedge (\neg(DesignType = Closed \vee DesignType = Open) \Rightarrow Spacer = null)$.

Data tables are simply a specific case of decision tables.

In (Deryck et al., 2019a), we defined the semantics of simple constraint tables (without quantification and functions) also as a conjunction of implications but without the final conjunct. The semantics of constraint tables and decision tables therefore differ only in the interpretation of incomplete tables: when no rows are applicable in decision tables, the output is forced to *null* (i.e., the implicit default value is *null*), while the output in constraint tables can take any value.

Now we extend this semantics to take variables and quantification into account.

Our first step is to define a function that maps cDMN expressions to terms: For the most part, this definition corresponds to that of Calvanese (Calvanese et al., 2019). However, we extend it to take into account the fact that certain expressions – which we call *variable expressions* – must be translated to FO variables. There are three kinds of variable expressions. We now define a mapping ν that maps each of these three kinds of cDMN variable expressions to a typed FO variable x of type T, which we denote as $x[T]$:

- The name T of a type is a variable expression. We define $\nu(T) = x_T[T]$, with x_T

a new variable of type T.

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

Given this function ν, we now define the following mapping $t_\nu(\cdot)$ of cDMN expressions to terms.

- For a constant c, $t_\nu(c) = c$; similarly, for an integer or floating point number n, $t_\nu(n) = n$.
- For an arithmetic expression e of the form $e_1 \theta e_2$ with $\theta \in \{+, -, *, /\}$, we define $t_\nu(e) = t_\nu(e_1)\ \theta\ t_\nu(e_2)$.
- For a variable expression v, we define $t_\nu(v) = \nu(v)$.
- For a function expression, i.e. "*Function* of arg_1 and ... and arg_n": $t_\nu(X) = Function(t_\nu(arg_1), ..., t_\nu(arg_n))$.

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

- If c is of the form "θe" with θ one of the relational operators $\{\leq, \geq, =, \neq\}$, then $F_{ij}(x)$ is the formula $x\ \theta\ t(e)$.
- If c is of the form *Not* e, then F_{ij} is $x \neq t(e)$.
- If c is a list $e_1, \ldots, e_n$, then F_{ij} is $x = t(e_1) \vee \ldots \vee x = t(e_n)$. As a special case, if c consists of a single expression e, then F_{ij} is $x = t(e)$.
- If c is a range, e.g. $[e_1, e_2)$, then F_{ij} is $x \geq t(e_1) \wedge x < t(e_2)$.

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

$$\underset{l \in V}{\forall} \nu(H_l) : \bigwedge_{i \in R} \left(\bigwedge_{j \in I} F_{ij}\big(t_\nu(H_j)\big) \Rightarrow \bigwedge_{k \in O} F_{ik}\big(t_\nu(H_k)\big) \right) \tag{28}$$

For example, in the the first row of Figure 16a,

- $\nu(H_1) = x[Component]$ and $t_\nu(H_1) = x$;
- $t_\nu(H_2) = IsUsed(t_\nu(H_1)) = IsUsed(x)$;
- $t_\nu(H_3) = Temperature$ and $t_\nu(F_{3,1})$ equals the formula $Temperature \geq MinTemp(Material(t_\nu(H_1)))$.

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

Figure 17: Decision table for dimensions.

Combining this leads to the FO(·) formula:

$$\forall x[Component] : IsUsed(x) \Rightarrow Temperature \geq MinTemp(Material(x)). \qquad (29)$$

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

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

7.6. Implementation

The previous section defines the semantics of cDMN by translating it FO(·). A practical solver has been implemented using the IDP-system. The specifics of the implementation can be found in (Vandevelde & Aerts, 2020).

7.7. Running Example

In this section we discuss the cDMN solution for the running example introduced in Section 3. The cDMN notation represents constraints in a table format. Consequently, the knowledge in the cDMN model will be almost identical to that of the constraint model discussed in Section 5.3.

7.7.1. Dimensions

Because we transform cDMN tables into FO(·) formulas that we give to IDP, our system can propagate the value of any arbitrary subset of variables in such a table to the other variables. In other words, the distinction between "input" and "output" variables may still be a useful way for domain experts to think about the structure of a table, but no longer has implications for how the table can be used by the system. Consequently, the relation between component dimensions can easily be represented in a decision table with a single row (Figure 17).

7.7.2. Design Type and Spacer

The constraints considering *design type* and *spacer* described in Section 5.3 can be easily put into two constraint tables with independent concerns, one for the concept of *Release Pressure* in Figure 18a and one for the use of a *Spacer* in Figure 18b. The

Release pressure			
E*	**Design Type**	**Position**	**Release Pressure**
1	Open	-	True
2	Closed	-	False
3	-	Pressure Accumulating	True

(a) Constraints related to Release Pressure.

Spacer is Used					
E*	**Design Type**	**Position**	**Back Pressure**	**Pressure**	**Spacer is Used**
1	Closed	-	-	-	False
2	-	Bi Directional	-	-	True
3	-	-	True	> 150	True

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

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

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

Figure 18: Constraint tables for the selection of Design Type and Use Component.

formulation of knowledge in constraint tables leaves considerable freedom in selecting a design and spacer. Section 7.7.5 discusses how this is handled.

Note that the constraint table in Figure 18a contains both descriptive (e.g., open designs can release pressure) and prescriptive information (e.g., in pressure accumulating positions, pressure should be released), as discussed in Section 5.1.

7.7.3. Materials

Figure 19 specifies for each of the components which materials can be used, e.g., the body can only be constructed from material M1, M2 or M3. Of course, this constraint is only relevant if the component is used in the product. When this is not the case, i.e., a component is not used, its material is set to null. The last line in Figure 19 states the additional constraint that the body of a closed design cannot be made of material M2.

In addition to these general constraints on the materials of all components, the materials each have a minimum/maximum temperature. Those temperatures are listed in a data table (Figure 16b). The "Temperature Constraints" table in Figure 16a states that, for each component that is actually used in the design, a material must be selected such that the operating temperature falls within the temperature range of this material.

As we discussed in Section 5.3, these constraints still leave considerable freedom in selecting a material. We will discuss how to prioritize certain materials in Section 7.7.5.

7.7.4. Shrinkage check

The decision tables in Figure 20 describe how to determine the *Shrinkage Load* and *Spring Strength* and state the constraint that the former should not exceed the latter. Note that this is a more intuitive way of representing the knowledge in Figure 7.

It enables us to differentiate between the constraint (i.e., *Shrinkage Load* should not exceed *Spring Strength*), and how to determine these values based on other design

Component Materials				
E*	**Component**	**Component is Used**	**Design Type**	**Material of Component**
1	Body	True	-	M1, M2, M3
2	Spring , Spacer	True	-	M1, M3, M5
3	-	False	-	null
4	Body	True	Closed	Not(M2)

Figure 19: General constraints on components' materials.

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

(a) Calculation of Shrinkage Load.

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

(b) Decision table to define Spring Strength.

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

(c) Constraint table for the shrinkage check.

Figure 20: Constraint approach to determine whether the shrinkage load can be handled.

properties. As discussed in Section 4.2, a work-around is needed in DMN to obtain a similar result, i.e., although the *Spring Strength* is a property of a spring with a certain thickness, we actually needed to make *Spring Thickness* an output of the DMN table that checks shrinkage, for lack of better solutions. As mentioned in Section 5.7, the *Shrinkage Load* and *Spring Strength* are also influenced by the *Design Type*. Because of cDMN's constraint-based nature, a switch from an open to a closed design will be automatically considered if the *Shrinkage Check* cannot be satisfied with an open design. We were not able to achieve this behaviour using standard DMN.

7.7.5. Preferences

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

the most reliable design. Such global optimization is not possible in regular DMN.

Figure 21 shows how an optimal design can be selected, based on a number of parameters. The underlying reasons behind these tables are identical to those defined in the *preferences* in Section 5.3.1.

Design Selection. We select a number of examples to highlight the alignment between the DMN tables in Section 4.2 and the cDMN model.

When both open and closed designs can be used, the leak tightness of closed designs is superior (Figure 21d). The cDMN model therefore selects a closed design when maximizing the optimization term, this is indeed what also happens in the DMN table in Figure 5a.

Thick springs have a higher cost than standard springs, as can be seen in Figure 21b, so they will only be selected when necessary to cope with a high shrinkage load.

Figure 21c states that the durability of a design is gravely affected when no spacer is used in a high back pressure (≥ 100) environment. In such circumstances, this design will never be used if a spacer design is available. This behaviour can also be found in row 4 and 5 of Figure 5a, where a closed design without spacer is only used in low temperature (i.e., high shrinkage) circumstances.

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

Materials. Because of the penalty for using locally unavailable materials (Figure 21e), the cDMN model never selects material *M4*. When this material would be available again, the adaptation to the cDMN model is trivial (i.e changing the availability in the Material Properties data table), while the DMN model would be more difficult to update.

When the temperature is between -50 and 100 degrees, all materials could be used for the spring. When taking into account the cost of each of the materials, specified in Figure 21a, cDMN selects the cheapest possible material, i.e., *M5*. This is indeed also the material selected by DMN in Figure 6b.

7.8. Discussion

We compare the cDMN model of our running example with the FO($\cdot$) representation presented in Section 5.3. A first conclusion is that the proposed notation is expressive enough to represent the knowledge in this running example, i.e., all knowledge fits well within this table-based notation. Second, the readability of the knowledge model considerably improves by expressing the knowledge in the cDMN language. Not only does the cDMN notation allow the decision logic to be expressed in a readable table-based notation, it also allows data to be expressed in a straightforward user-friendly manner (i.e., typically data is already available in similar tables).

We also compare of the DMN model discussed in Section 4.2 and the cDMN model in this section. A first observation is that the cDMN model is considerably larger than the standard DMN model, i.e., the DMN model consists of only nine decision tables, whereas the cDMN model consists of ten tables that determine the physical constraints

Data Table: Material Properties			
	Material	**Cost of Material**	**Material is Locally Available**
1	M1	8	True
2	M2	5	True
3	M3	10	True
4	M5	4	True
5	M4	7	False

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

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

(b) Define total cost of design.

Durability				
C+	**Spacer is used**	**Back Pressure**	**Pressure**	**Durability**
1	False	True	< 100	-2
2	False	True	≥ 100	-100
3	-	-	-	$-1/5\times$ Spring Strength

(c) Define durability of design.

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

(d) Define leak tightness of design.

Availability				
C+	**Component**	**Component is used**	**Material of Component is Locally Available**	**Availability**
1	-	True	False	-100

(e) Define availability penalty.

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

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

Figure 21: Preferences in cDMN.

and an additional six tables to express the possible preferences. The larger size of the cDMN model is compensated for in terms of practicality:

- 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 solution is selected.
- 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.
- As a result of introducing multiple optimization criteria, the same model can be used when, for a specific request, the priorities deviate from the standard priorities. In regular DMN, a new model would have to be created for each possible optimization function.

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

8. Related Work

Product Configuration. There exists a vast body of research on product configuration systems, which are typically defined as systems to automatically construct a design from a set of pre-defined components, considering several constraints and some optimisation criteria (Brown, 1998). The advantages of using product configurators are well documented and include improvement of lead time (Haug et al., 2011, Forza & Salvador, 2002), sales quotation time (Hvam et al., 2004), product quality and associated costs, sales, and supplier communication (Haug et al., 2019).

A thorough literature review on product configuration was performed by (Zhang, 2014). 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 recent work of Shafiee et al. (Shafiee et al., 2018) who developed a framework for the knowledge acquisition in configuration projects. They propose a four-step framework consisting of determining the scope of the application, knowledge acquisition, modelling and knowledge validation and documentation and maintenance. Although these elements are also present in our project, they are not separated into distinct steps because of our focus on the creation of (c)DMN models together with the product engineers. The act of creating the models encompasses both the knowledge acquisition and modelling phase, while the resulting tables can easily be validated and maintained by (other) product engineers in the company, and form a complete knowledge base. A second difference is that, in the

framework of Shafiee, knowledge engineers structure, initiate and execute the different phases, while in our approach the product engineers themselves play a central role at every stage.

A second aspect which according to (Zhang, 2014) has not yet received much attention is the ability to suggest new designs. The majority of existing product configuration approaches focus on selecting the most appropriate option among a fixed range of possibilities. By contrast, our constraint-based approach is also able to provide useful information to the engineers in cases that fall outside the scope of existing solutions.

Third, (Zhang, 2014) 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 (Denecker, 2008), in which different logical inference methods can be applied to the same knowledge base in order to implement different functionalities. Indeed, in Section 6, we describe how the AutoConfig tool can be used in an interactive way, which, amongst other functionalities, contains the ability to explain the system's behaviour. Moreover, the task of reconfiguration has already been considered in the context of this system (Vlaeminck et al., 2009, Wittocx et al., 2009). The IDP system therefore provides a suitable framework to implement these different tasks.

DMN. The use of DMN has become widespread in business and is well-documented in academic research since its inception in 2015 (Kluza et al., 2019). Several papers describe the use of DMN in industrial and manufacturing environments. E.g.; the Composelector H2020 project created a platform aimed to support the multi-scale selection of composite material (Belouettar et al., 2018). DMN is used to refer the decision maker to the correct component of the platform. In another example, Peinl and Perak (Peinl & Perak, 2019) describe how they use BPMN and DMN for the customization of Manufacturing Execution Systems (MES). They started from the observation that academic literature suggests to use as much standardized software as possible, but that this is difficult due to the industry-specific needs and the evolution towards lot-size one. Therefore, they propose HiCuMES, a highly modular solution, in which they use DMN to model different product types that require different process steps. In contrast to our application, DMN is not used to automate actual engineering decisions in either application.

Interactivity is an important quality of a decision support system. As DMN tables are typically static and not designed for interaction, real life applications have had to solve this in different ways. Sooter (Sooter et al., 2019) describes a combined BPMN/DMN approach to model an extended decision process in which user (inter-)actions are modelled in a BPMN model to which the DMN model is attached. In other words, interaction is only possible in certain fixed steps of the decision process. Batoulis et al. (Batoulis et al., 2015) describe how dynamic decision making can be supported by dynamically updating decision tables based on the current availability of needed resources. As already discussed, we propose a different approach in which DMN and cDMN tables can be used as a declarative knowledge base, to which different forms of logical inference can be applied to produce the desired interactive behaviour in the IDP

system.

Apart from the interactivity issues, DMN decision tables are also often not expressive enough to model real-life cases (Calvanese et al., 2019, Deryck et al., 2019a, Car, 2018). The DMN standard does propose the Friendly Enough Expression Language (FEEL), that is able to adress more complex situation than the S-FEEL language that we used (OMG, 2019). However, as it is a programming language with its own syntax, it cannot be used directly by domain experts, hence limiting its usefulness.

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

Other approaches allow DMN to be extended by more declarative representations. For instance, the aforementioned OpenRules also offers an interface to a constraint solver (Feldman, 2011), while (Calvanese et al., 2019) allows DMN to be enriched with domain knowledge expressed in Description Logic. The latter paper demonstrates its approach by means of a case study of port clearance for cargo ships. Here, the clearance decision itself can be modelled with traditional DMN, but the ontology of cargo ship types and their features are modelled in Description Logic.

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

9. Conclusions and future work

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

This project's main challenge is that there are two potentially contradictory requirements. On the one hand, a flexible and powerful knowledge representation method is needed to provide support to the engineers even in circumstances that fall outside of their designs' usual scope. On the other hand, the company needs a system that can deterministically propose standard designs, to assist the people in the sales department. Moreover, for the creation of both applications, engineers needed to be closely involved, as they posess the necessary product knowledge to create and help maintain the knowledge base.

Taking these requirements into account, we modelled knowledge in both a deterministic and constraint-based way, using the DMN standard for the former and FO(.) in the IDP system for the latter.

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

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

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

In future work, we will be looking into a more standardised approach to represent missing data (e.g., when material properties are not known). Moreover, it would be interesting to investigate whether historical data can be used to identify additional constraints that can be used to supplement those in the knowledge base. An additional possibility is to extract knowledge from the general constraint-based knowledge base, to automatically create or update the deterministic procedure of a certain sub-application, and to keep knowledge in these two systems up-to-date. Currently, the constructed knowledge base is maintained manually, but it could be useful to find a way to assist the knowledge base maintainer in this process.

Finally, the IDP-system is not able to perform floating point calculations. Therefore, we are looking into the possibility to port the system to a back-end that can handle these calculations.

Acknowledgements

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

References

Aerts, B., Vandevelde, S., & Vennekens, J. (2020). Tackling the dmn challenges with cdmn: A tight integration of DMN and constraint reasoning. In *Proceedings of RuleML+RR 2020*, vol. abs/2005.09998, (pp. 23–38). Gutiérrez Basulto, Victor, Springer International Publishing.

Aerts, B., & Vennekens, J. (2018). Application of logic-based methods to machine component design. In *Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018)*, (pp. 13:1–13:15).

Alviano, M., Calimeri, F., Charwat, G., Dao-Tran, M., Dodaro, C., Ianni, G., Krennwallner, T., Kronegger, M., Oetsch, J., Pfandler, A., & Others (2013). The fourth answer set programming competition: Preliminary report. In *Logic Programming and Nonmonotonic Reasoning*, (pp. 42–53). Springer.

Batoulis, K., Baumgraß, A., Herzberg, N., & Weske, M. (2015). Enabling dynamic decision making in business processes with DMN. In *International Conference on Business Process Management*, (pp. 418–431). Springer.

Belouettar, S., Kavka, C., Patzak, B., Koelman, H., Rauchs, G., Giunta, G., Madeo, A., Pricl, S., & Daouadji, A. (2018). Integration of material and process modelling in a business decision support system: Case of composelector h2020 project. *Composite Structures*, *204*, 778–790.

Biard, T., Le Mauff, A., Bigand, M., & Bourey, J.-P. (2015). Separation of decision modeling from business process modeling using new decision model and notation (DMN) for automating operational decision-making. In *Working Conference on Virtual Enterprises*, (pp. 489–496). Springer.

Bogaerts, B., Bruynooghe, M., Janssens, G., & Denecker, M. (2018). Predicate logic as a modelling language: The IDP system. *arXiv.org*.
URL `http://search.proquest.com/docview/2071739773/`

Brown, D. C. (1998). Defining configuring. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *12*(4), 301–305.

Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M., & Verwer, S. (2015). Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with idp3. *Theory and Practice of Logic Programming*, *15*(6), 783.

Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F. M., Montali, M., & Teinemaa, I. (2016). Semantics and analysis of DMN decision tables. In *International Conference on Business Process Management*, (pp. 217–233). Springer.

Calvanese, D., Montali, M., Dumas, M., & Maggi, F. M. (2019). Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge. *Theory and Practice of Logic Programming*, *19*(4), 536–573.

Car, N. J. (2018). Using decision models to enable better irrigation decision support systems. *Computers and Electronics in Agriculture*, *152*, 290–301.
URL `http://www.sciencedirect.com/science/article/pii/S0168169917313595`

Dasseville, I., Janssens, L., Janssens, G., Vanthienen, J., & Denecker, M. (2016). Combining DMN and the knowledge base paradigm for flexible decision enactment. In T. Athan, A. Giurca, R. Grütter, M. Proctor, K. Teymourian, & W. Van Woensel (Eds.) *Supplementary Proceedings of the RuleML 2016 Challenge, RuleML, Stony Brook, 6-9 July 2016*. CEUR-WS.org.
URL `https://lirias.kuleuven.be/handle/123456789/546123`

Denecker, M. (2008). Building a knowledge base system for an integration of logic programming and classical logic. vol. 5366, (pp. 71–76). Springer.
URL https://lirias.kuleuven.be/bitstream/123456789/235155/1/paper.pdf

Deryck, M., Aerts, B., & Vennekens, J. (2019a). Adding constraint tables to the DMN standard: Preliminary results. In *International Joint Conference on Rules and Reasoning*, (pp. 171–179). Springer.

Deryck, M., Devriendt, J., Marynissen, S., & Vennekens, J. (2019b). Legislation in the knowledge base paradigm: interactive decision enactment for registration duties. In *proceedings of the 13 the IEEE Conference on Semantic Computing*, (pp. 174–177). IEEE.

Feldman, J. (2011). Representing and solving rule-based decision models with constraint solvers. *7018*, 208–221.

Forza, C., & Salvador, F. (2002). Product configuration and inter-firm co-ordination: An innovative solution from a small manufacturing enterprise. *Computers in Industry*, *49*(1), 37–46.

Hasić, F., Corea, C., Blatt, J., Delfmann, P., & Serral, E. (2020). A tool for the verification of decision model and notation (DMN) models. In *Research Challenges in Information Science*, Lecture Notes in Business Information Processing, (pp. 536–542). Cham: Springer International Publishing.

Haug, A., Hvam, L., & Mortensen, N. H. (2011). The impact of product configurators on lead times in engineering-oriented companies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM*, *25*(2), 197–206.

Haug, A., Shafiee, S., & Hvam, L. (2019). The costs and benefits of product configuration projects in engineer-to-order companies. *Computers in Industry*, *105*, 133 – 142.
URL http://www.sciencedirect.com/science/article/pii/S0166361518304263

Hvam, L., Malis, M., Hansen, B., & Riis, J. (2004). Reengineering of the quotation process: application of knowledge based systems. *Business Process Management Journal*, *10*(2), 200–213.

Kluza, K., Adrian, W. T., Wisniewski, P., & Ligkeza, A. (2019). Understanding decision model and notation: DMN research directions and trends. In C. Douligeris, D. Karagiannis, & D. Apostolou (Eds.) *Knowledge Science, Engineering and Management*, (pp. 787–795). Cham: Springer International Publishing.

Mertens, S., Gailly, F., & Poels, G. (2015). Enhancing declarative process models with DMN decision logic. In *International Conference on Enterprise, Business-Process and Information Systems Modeling*, (pp. 151–165). Springer.

Mitchell, D., Ternovska, E., Hach, F., & Mohebali, R. (2006). Model expansion as a framework for modelling and solving search problems. Tech. rep., Technical Report TR 2006-24, School of Computing Science, Simon Fraser University.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). Minizinc: Towards a standard CP modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, vol. 4741 of *LNCS*, (p. 529).

OMG (2019). Decision model and notation.
URL `http://www.omg.org/spec/DMN/`

Peinl, R., & Perak, O. (2019). BPMN and DMN for easy customizing of manufacturing execution systems. In C. Di Francescomarino, R. Dijkman, & U. Zdun (Eds.) *Business Process Management Workshops*, (pp. 441–452). Cham: Springer International Publishing.

Shafiee, S., Kristjansdottir, K., Hvam, L., & Forza, C. (2018). How to scope configuration projects and manage the knowledge they require. *Journal of knowledge management*, *22*(5), 982–1014.

Shafiee, S., Wautelet, Y., Hvam, L., Sandrin, E., & Forza, C. (2020). Scrum versus rational unified process in facing the main challenges of product configuration systems development. *Journal of Systems and Software*, *170*, 110732.
URL `http://www.sciencedirect.com/science/article/pii/S0164121220301643`

Sooter, L. J., Hasley, S., Lario, R., Rubin, K. S., & Hasić, F. (2019). Modeling a Clinical Pathway for Contraception. *Applied clinical informatics*, *10*(5), 935–943.
URL `https://europepmc.org/articles/PMC6924335`

Van Daele, D., Decleyre, N., Dubois, H., & Meert, W. (2019). An automated engineering assistant: Learning parsers for technical drawings. *arXiv preprint arXiv:1909.08552*.

Vandevelde, S., & Aerts, B. (2020). cDMN documentation.
URL `https://cdmn.readthedocs.io/en/latest/`

Vlaeminck, H., Vennekens, J., & Denecker, M. (2009). 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*. ACM.
URL `https://lirias.kuleuven.be/handle/123456789/262013`

Wang, Y., Mo, D. Y., & Tseng, M. M. (2019). Relative preference-based product configurator design. *Procedia CIRP*, *83*, 575–578.

Wittocx, J., De Cat, B., & Denecker, M. (2009). Towards computing revised models for FO theories. In S. Abreu, & D. Seipel (Eds.) *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, (pp. 199–211). URL https://lirias.kuleuven.be/handle/123456789/242185

Wittocx, J., Mariën, M., & Denecker, M. (2008). The idp system: a model expansion system for an extension of classical logic. In *Proceedings of the 2nd Workshop on Logic and Search*, (pp. 153–165). ACCO; Leuven.

Zhang, L. L. (2014). Product configuration: A review of the state-of-the-art and future research. *International Journal of Production Research*, *52*(21), 6381–6398. URL http://dx.doi.org/10.1080/00207543.2014.942012