Application of Logic-Based Methods to Machine Component Design

Bram Aerts
EAVISE
Technology Campus De Nayer, KU Leuven, Belgium
b.aerts@kuleuven.be

Joost Vennekens
EAVISE
Technology Campus De Nayer, KU Leuven, Belgium
joost.vennekens@kuleuven.be

Abstract

This paper describes an application worked out in collaboration with a company that produces made-to-order machine components. The goal of the project is to develop a system that can support the company's engineers by automating parts of their component design process. We propose a knowledge extraction methodology based on the recent DMN (Decision Model and Notation) standard and compare a rule-based and a constraint-based method for representing the resulting knowledge. We study the advantages and disadvantages of both approaches in the context of the company's real-life application.

2012 ACM Subject Classification Applied computing → Engineering

Keywords and phrases Application, Expert Systems, Constraint Solving, Rule-based Systems, Decision Modelling, DMN, Product Configuration

Digital Object Identifier 10.4230/OASIcs.ICLP.2018.13

1 Introduction

This research is conducted in collaboration with a company that has engineering and manufacturing offices all over the world. To protect its trade secrets, the company wishes to remain anonymous and they have also requested that we avoid providing too much information about its products. In certain branches of its activities, the company specialises in producing made-to-order components, designed specifically to meet a customer's particular requirements. Like many such companies, it has significantly automated its manufacturing activities, but the design activities of its engineers are still performed "manually". That is to say, the engineers of course make use of computers to perform calculations or create 3D models of the components they design, but there is no software support for the crux of their activity, namely the actual design process itself. To perform this task, the engineers follow an *ad hoc* process, based on past experience, talks with their colleagues, their own preferences, etc.

This way of working is still common in industry. However, it has several downsides. First, the lack of standardisation means that different engineers at different locations may come up with different designs for the same set of requirements, some of which may be worse than others. Second, the company also depends to a large extent on the expertise of some of its key senior engineers. If these should suddenly leave the company, a great deal of the knowledge they have built up over the years would leave with them, significantly reducing

the efficacy of the engineering department. Finally, the lack of software support also means that – in particular, for less challenging design tasks – the engineers often have to spend time carrying out the same routine tasks, reducing their efficiency.

The goal of this research is to develop a system to assist the engineers in their design process. We focus specifically on the design of one particular type of component. This type of component consists of a number of different subparts, each of which exists in a number of different variants and sizes, and which can be produced from different kinds of materials. Customers request components for a specific set of requirements, including a temperature range under which the component should function, pressures the component should be able to withstand, the size that the component should have, etc. The engineers then decide which combination of subparts should be used, which variants of these subparts should be chosen, how big each subpart should be and out of which material it should be made. It is with this task that we want to assist them.

We follow a knowledge-based approach, in we represent the engineers' knowledge in a suitable formal language, and then apply logical inference to this representation in order to provide suggestions to the engineers. This approach starts with a knowledge extraction step in which a knowledge engineer works together with a number of domain experts, in this case the company's design engineers, in order to construct the formal model of their knowledge. Typically, this knowledge extraction is a challenging task, because the knowledge engineers are not familiar with the problem domain, while the domain experts are not familiar with the idea knowledge representation. Good communication between the parties is therefore very important.

In addition to providing automated support, the knowledge extraction process also has the benefit of producing a standardised formal description of the company's design knowledge, thereby eliminating personal preferences of each engineer, regional differences, out-dated habits, and of course human mistakes. For this process to be successful, we believe that it is crucial that the formal specification is not only executable, but that it is also understandable by the engineers. This helps to avoid misunderstandings and errors in the knowledge extraction process. Moreover, it will also allow the engineers to get a better understanding of what is going on inside the decision support system, it will help them to adopt and evaluate the standardised procedure, and it will allow the knowledge base to be maintained after completion of the project.

In [24], the ability to extract knowledge in a format readable by domain expert was identified as a weakness of current product configuration methods. In order to achieve our stated goals, we therefore propose a novel method, consisting of a two-step knowledge extraction methodology. First, we focus on representing the *decision process* that the engineers follow when making a new design. For this, we make use of the recent *Decision Model and Notation (DMN)* [13] standard, which has been developed with the specific aim of being usable by domain experts, without help from a knowledge engineer or software developer. Using an off-the-shelf implementation of the standard, such as that provided by the OpenRules system [14], this DMN model is already fully executable, which allows it to be used by the engineers and validated w.r.t. a batch of test cases.

As we will discuss below, the DMN model by itself is not expressive enough to achieve all of the project's goals. We therefore propose a second knowledge extraction step, in which the DMN model is further analysed together with the design engineers. Having the DMN model already available in this step provides a way of focusing the discussion, ensuring that all the relevant questions end up being discussed, and avoiding misunderstandings. The result of

this second step is a logical specification, written in classical first-order logic – which can be used by an automated reasoning system – in our case the IDP system [3]. This specification can then be validated by comparing its conclusions to those of the original DMN models.

In the following sections, we first provide some more details on the context and goals of the project. We then discuss the first step of the knowledge extraction methodology, using DMN, together with its implementation and limitations. We then present the second step, using the IDP system, again also discussing implementation and limitations. We discuss the validation efforts that were made and finally also related work.

2 Problem Description

The company designs and produces components based on specific customer requests. These customers typically are engineerings from other production companies, who want a specific part to be manufactured according to a detailed set of requirements. In contrast to typical configuration problems, understanding and explicitating the customers' needs is therefore not an issue in this application.

Incoming requests are initially handled by the sales staff. If the customer's requirements can be met by one of the companies standard solutions, the sales staff autonomously handles the request. They are supported in this by a Visual Basic tool that inspects a Microsoft Access database to select the appropriate standard design for a particular request. Requests that fall outside the scope of this tool are forwarded to the engineering department. Here, one of the engineers analyses the requirements and proposes a suitable component design. A distinction is made between requests that fall within known application areas and those that do not. Handling the first kind of requests is a routine job for the engineers and they always follow roughly the same procedure when doing so. The second kind of requests are more challenging and may require a significant amount of creativity from the engineers.

Our project has three main goals. First, the company has noted that is quite difficult and time-consuming to extend the scope of the tool that is used by the sales staff and they are looking for a more maintainable solution. Second, the "routine" work done by the engineers for known applications should be standardised and automated as much as possible. Third, the company also wishes to develop a decision support system that the engineers can use when handling the more challenging requests.

3 Knowledge extraction of the design process

The engineers have a "standard" decision process that they use to handle routine requests. However, this process is not explicitly standardised and different engineers at different locations may do certain things somewhat differently. To fully standardize this process and to be able to automate it, the engineers' detailed technical knowledge needs to be represented in a formal and structured manner. This section describes the knowledge extraction methodology that we have followed.

Because the design process had not yet been internally standardised, we chose to start from a series of brainstorming workshops with all of the involved parties. Each workshop takes a couple of days and results in an initial representation of the design process for a specifically delineated application area. The involved parties are a number of design engineers (representing each of the locations worldwide that are involved in the particular application area), a manager and one external knowledge engineer to guide the workshop. This approach offers a number of advantages.

- Since multiple participants are involved, we do not blindly adopt the approach of one engineer or one particular location.
- The face-to-face time allows intensive discussion about why certain decisions are taken, which is often necessary when different engineers are used to follow different approaches.
- During one multiple-day workshop, all parties focus solely on one specific application, which helps to keep the discussion focused.
- The knowledge engineer not only helps with technical issues concerning the representation, but he also assists the engineers in clarifying their design process: as a non-expert in the domain, he is able to ask "trivial" questions that help to ensure that all the engineers are on the same page and that nothing is being overlooked.

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

To support this knowledge extraction process, we need a notation that allows all aspects of the decision process to be expressed. In addition, the notation should not only be readable by the knowledge engineer, but also by the domain experts, who have no background in computer science or logic. This will allow the notation to be used as an effective communication tool throughout the brainstorming workshops and will also give the domain experts confidence in the correctness of the automated system. After surveying the different possibilities, we have decided to use the DMN standard that is explained in the following section.

3.1 The Decision Model and Notation (DMN)

The Decision Model and Notation (DMN) is a relative new standard [13], which is best known for also being responsible for the widely used UML standard. This standard was developed specifically for describing and modeling repeatable decision processes. In addition, it is especially designed to be usable by "business users", without involvement of IT personnel. These two properties make it uniquely well-suited for our purposes. In addition, as an open standard from a well-known organisation, it enjoys tool-support from multiple vendors, which means that it can be adopted without running the risk of vendor lock-in.

In general, a DMN model consists of two components. The first is a Decision Requirement Diagram (DRD). This is a tree-like graph which specifies dependencies between different (sub-) decisions. Figure 1 displays a fragment of the complete DRD representing the decision procedure used in our application.

The other part of a DMN model consists a number of in-depth decision tables, one for each decision in the DRD. An example can be found in Table 1. The purpose of this table is to decide whether the chosen design should contain a *wiper*, a bent piece of plastic that protects the component from environmental factors, such as dirt or reverse pressure (i.e., pressure from the outside to the inside, instead of the other way around). Each column of such a table corresponds to either an input variable (*Dirty Environment* and *Reverse Pressure*, in this case) or an output variable (*Wiper*). In this example, all variables are boolean, but in general DMN also allows other data types. A row in a decision table specifies that if the row is applicable (i.e., all of the input variables satisfy the conditions given by this row) then all of the output variables must have the values given by this row. For instance, the first row of Table 1 states that a wiper must be used whenever the environment is dirty (regardless of

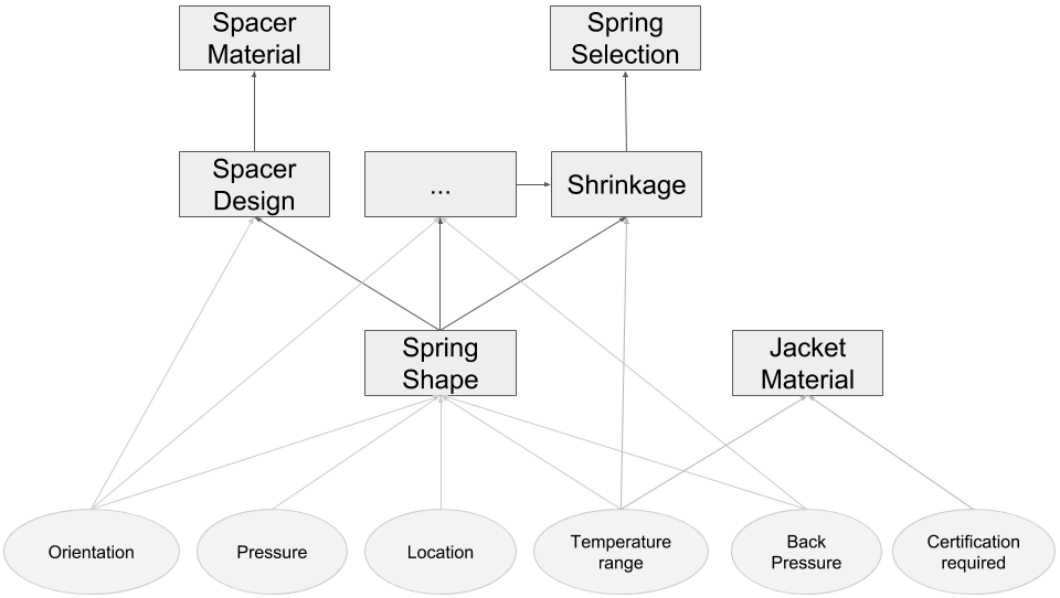

Figure 1 Fragment of the Decision Requirement Diagram.

Table 1 Decision table describing whether or not to use a wiper.

Any	Input		Output
	Dirty Environment	Reverse Pressure	Wiper
1	True	-	True
2	-	True	True
3	False	False	False

whether there is reverse pressure); the second row states that if there is reverse pressure, a wiper must also be used; finally, the third row states that if the environment is not dirty and there is no reverse pressure, a wiper should not be used.

The entries in the table are written in a syntax called the Friendly Enough Expression Language (FEEL), which is also part of the DMN standard. In addition to simple values (as used in Table 1), FEEL also allows numerical comparisons, ranges of values and calculations to be expressed.

If multiple rows in a table might be applicable for some combination of input values, then the table's so-called *hit policy* determines how this should be handled. Table 1 has the hit policy *Any*, as can be seen in its upper left cell. This means that different rows may be applicable for a given input (e.g., the first two rows are applicable in a dirty environment with reverse pressure), but that all applicable rows have the same output, so that it does not matter which row is applied. Other hit policies are *Unique* (only one row may be applicable) and *First* (when multiple rows are applicable, only the top one is considered). In addition, there are also *multiple hit policies* that allow, e.g., the output of all applicable rows to be gathered into a list.

Another, more advanced example is the following. In the design of a component, a spring is used to keep it in place. The type of this spring is determined by two decision nodes in the DRD. First, the general shape of spring is determined (whether to use a stiffer closed spring or a weaker open spring). This influences the overall form of the design. Later, the

Table 2 Decision table for *Spring Shape*.

First	Input					Output
	Orientation	Reverse Pressure	Location	Pressure	Temperature	Spring Shape
1	Radial	True	Pressure Accumulating	-	-	Open
2			Bi-directional	-	-	Open
3			-	≤ 100	-	Open
4	-	-	-	-	-	Closed

Table 3 Decision table for *Use Of Spacer*.

Unique	Input	Output
	Spring Shape	Use Spacer
1	Closed	False
2	Open	True

specific spring is selected, based on how much the component would shrink in the given circumstances. Table 2 shows how the general shape of the spring is decided, based on the reverse pressure and various other inputs.

Another part of the design is a spacer, whose purpose is to keep the component in place, even when there is a high pressure from the backside of the seal. Based on the spring shape, the need for a spacer is decided in Table 3.

3.2 Results

Following the methodology outlined above, we have extracted the knowledge of the routine design process in six different application fields. A total of 75 decision tables were constructed. In each of the applications, one or two tables were pure data tables, consisting of all numerical data for dimensioning the component. Since the discussed applications are more or less similar, some of the already constructed data and decision tables from one application could be reused in another. The extracted tables had an average size of approximately 5 rows and 3 input conditions.

Each workshop started with a brief introduction to DMN, after which the knowledge engineer started to guide the domain experts through the modelling process. We typically started by constructing a DRD to get a general overview of the structure of the design process, and then proceeded to construct detailed decision tables for each of the decisions. The company's engineers found the DMN format quite intuitive and after some initial questions, they were typically able to easily interpret and reason about the knowledge in the tables. Our experiences therefore indeed confirm that DMN's readability for domain experts is a big advantage of this standard.

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

Overall, the DMN representation seemed to fit well with the engineers' own way of thinking about their design process. However, there were some exceptions. In a few limited cases, the engineers themselves do not follow a strict bottom-up decision procedure when making their design. For instance, in certain circumstances, it is necessary to ensure that the component stays in place. This can be done by using a stiffer spring than usual to prevent

the component from sliding in the wrong direction. Adding a spacer and keeping the normal shape of spring is the preferred approach, but this is not always feasible. In particular, in cold circumstances, the component may shrink to such an extent that the normal spring would fail. However, to know whether this is the case, the shrinkage of the component has to be computed. Because this depends on the materials being used and the precise layout of the different parts of the component, this computation can only be done at the very end of the design process. Therefore, what the engineers currently do is they assume that the spacer option will work, completely design the component based on this assumption, compute the shrinkage and then backtrack over their initial choice if it turns out that the shrinkage is too big. Such a "guess and check" procedure cannot be elegantly represented in DMN. In Section 4 we discuss the work-around that we have used for this.

In general, we perceived the use of a formal representation in the workshop as a significant added value. The precision of the notation allowed us to quickly detect inconsistencies and missing cases in the information that the domain experts were providing. In addition, once they had gotten used to the notation, also the design engineers themselves started to notice flaws in the decision tables, such as implementation mistakes from our side or previously unnoticed exceptions in their own design process. Towards the end of a workshop, the design engineers were comfortable enough with the notation that we could leave certain decision tables to be constructed as "homework" after the end of the workshop.

Based on our experiences, we are confident that the design engineers will be able to maintain the existing decision tables and, with a bit more experience, would be able to construct additional DMN models for new application areas.

4 Direct implementation of the design process

DMN is designed to be a fully executable specification and is currently supported by a number of different tools, both commercial and open source. By providing it with the constructed DMN tables, we have implemented an automated design system in the OpenRules [14] system, currently for two of the six application areas for which the DMN knowledge extraction has been performed.

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

First, as mentioned in Section 3.2, a few aspects of the design process do not fit readily into the DMN model. Currently, we have worked around this problem by an "err on the side of safety" approach: for the example given Section 3.2, the engineers have determined a set of parameters within which it is always safe to use the preferred solution of adding a spacer; whenever the input falls outside of this safe range, the alternative option of using a stiffer spring is always chosen. While this solution is suboptimal (in the sense that sometimes a stiffer spring is used when the combination of a weaker spring and a spacer would have sufficed), it avoids the risk of suggesting faulty designs in a way that does not introduce complicated decision structures, which would reduce the legibility of the DMN model.

Second, the DMN representation forces one to mix different kinds of knowledge within a single table, which reduces the maintainability. For instance, Table 2 is based on both physical constraints and preferences of the company. However, the actual constraints and preferences cannot be deduced from this table alone. For instance, the decisions could be explained in any of the following three ways:

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

Now, suppose that a supplier changes the price of the closed spring. This will have an impact on which shape of spring is preferred, but it is impossible to judge the impact of this change on Table 2, without knowing the underlying reason for why this table is as it is. A representation that separates preferences from constraints would not have this problem.

Third, all of the currently available DMN rule engines support only a single inference task, namely that of computing the "output" decision variables given values for all the input variables. In a system that is used interactively by a design engineer, however, we may also envisage other useful inference tasks. For instance, after filling out only a subset of the input variables, the engineer may be interested in knowing whether a design with a closed spring is still possible. Or, in discussions with a customer, he may interested in knowing which values of the input variables would have allowed such a spring to be used if one cannot be used now.

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

Finally, the entire DMN approach of course assumes that there *is* a decision procedure to model. If we want to develop a system that could provide some assistance to engineers in those challenging new application areas where they themselves do no yet know how precisely a new design should be made, then there is no decision procedure and the DMN approach will be of no use at all.

5 A Constraint-Based Approach

As discussed in the previous section, we cannot hope to achieve all of our stated goals by an approach in which we simply use a direct implementation of the design procedure as the engineers follow it. We will need to take into account also 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 $< 100°C$) 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 constraints provide more information than is explicitly present in the design procedure, because they also explain *why* certain designs are impossible. Therefore, it is not possible to automatically deduce these constraints from the design procedure. Instead, coming up with them requires additional discussions with the design engineers.

To illustrate the constraint-based approach, we return to the running examples of Section 3.1. First, we consider Table 1. The engineers explain the contents of this table as follows: they prefer not to include a wiper unless one is necessary, and a wiper is required

to cope with either reverse pressure or a dirty environment. In other words, this table can be explained as the combination of a preference for not having a wiper, together with two constraints: $ReversePressure \Rightarrow Wiper$ and $DirtyEnvironment \Rightarrow Wiper$.

The underlying reasons for Tables 2 and 3 are more complex. Discussions with the engineers have revealed that these tables can be explained as follows:

1. Only open springs are able to release reverse pressure.
   $SpringShape = "Open" \Leftrightarrow AbleToReleaseBP$.
2. It is impossible to use a spacer in combination with a closed spring.
   $SpringShape = "Closed" \Rightarrow SpacerDesign = "null"$.
3. When the component should be placed in a pressure accumulating location, it should be able to release reverse pressure.
   $Location = "PressureAccumulating" \Rightarrow AbleToReleaseBP$.
4. A spacer is needed (in radial applications) if the reverse pressure is bigger than 100 bar.
   $ReversePressure \wedge Pclass > 100 \Rightarrow SpacerDesign \neq "null"$.
5. In the bi-directional location, the component tends to move back and forth excessively, so in order to avoid damage, a spacer is always needed.
   $Location = "Bi-directional" \Rightarrow SpacerDesign \neq "null"$.
6. Lastly, closed springs tend to be cheaper and outperform open springs, so they are the preferred type of spring.

Notice that 1–5 are constraints, while 6 is a preference.

The first line in Table 2 is a result of combining constraint 1 and 3. The component should be able to release reverse pressure and since closed spring designs cannot do that, an open spring design is the only option. The second row is a combination of constraint 2 and constraint 5. In the "Bi-directional" location a spacer is always needed, and since it is impossible to have a spacer in closed spring designs, the only remaining possibility is to go for an open spring design. Analogously, the third line in the decision procedure can be obtained from combining constraint 4 and 2. In all other situations, both closed and open spring designs are possible, but closed designs are preferred, which explains the last row in the decision procedure.

5.1 Knowledge extraction of the physical constraints and preferences

In order to use the physical constraints, we must of course again first elicitate them from the design engineers. In our experience, it was difficult to do this directly. The engineers often did not know quite where to start and discussions tended to be chaotic and unstructured. For this reason, we have chosen to base the knowledge extraction of the constraints on the DMN models. We again organise a discussion with the engineers who were originally involved in the construction of these models and then go over each row of each table and ask them *why* this row produces that particular output. Unlike the workshops in which the DMN models are initially constructed, here it is less crucial to involve different engineers: even though different engineers may disagree on the best solution for a given problem, they tend to all 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 all of the relevant constraints will eventually be mentioned. Moreover, because the engineers know and understand the DMN model, there is never any confusion about which particular question is being discussed at any particular point in time.

To reduce the time investment required from the engineers, it is 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 (e.g., the "default" row at the bottom of Table 2 suggests that the closed spring is the preferred choice, with the other rows describing circumstances in which this preferred choice is not possible). In addition, general knowledge about how the components function or considerations that were mentioned during the workshops that constructed the DMN models may provide further clues. In practice, we have found that we can construct most of the constraints without help of the engineers and only need them to verify and help us revise our initial guesses.

Most of the decision tables can be discussed independently. However, certain constraints influence multiple tables. Section 5 handles a detailed example of this.

The preferences we have encountered so far have been quite simple: when a particular part exists in a number of different variants or can be made from a number of different materials, the engineers have been able to rank the variants/materials in an absolute order of preference, typically based on cost and reliability. There has been no need to handle more complex issues such as conditional preferences.

6 Implementation of a constraint-based approach

We have used the knowledge based IDP system [3] to implement a prototype of a constraint-based design system. IDP allows constraints to be expressed in a rich extension of classical first-order logic. Some examples of constraints used, are:

$$\forall s[Subpart] : SubpartUsed(s) \Rightarrow \exists\ 1\ m[Material] : Material(s, m).$$

This IDP formula states that for each subpart it holds that if the subpart is used, there exists exactly one material for that subpart.

$$sum\{s[Subpart] : SubpartUsed(s) \wedge Length(s, l) : l\} < AvailableSpace.$$

This formula states that the length of the component, computed as the sum of the lengths of all its subparts, must fit in the available space.

The IDP system offers a number of 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 [1], it was shown to be competitive with other state-of-the-art ASP systems, though typically somewhat slower than systems such as Clasp.

Our main reason for using IDP is its use of classical logic as an input language. This allows individual constraints to be represented in a modular way, which can typically be reasonably well explained to the company's design engineers without requiring much additional background. While the engineers would probably not be able to write down constraints correctly, they are able to read them pretty well. We suspect that for instance ASP specifications would have been harder for the engineers to read, due to the presence of non-classical connectives such as negation-as-failure. A second advantage of IDP is that it provides support for different logic inference tasks. Our current prototype only offers the functionality of generating design proposals, but IDP's different logic inference methods may prove useful if we would want to extend this to other functionalities in the future. This is one potential advantage that IDP offers over the use of constraint-programming languages such as MiniZinc [12].

Our input for IDP consists of six theories: one theory expresses the constraints about the general design of the component; another describes the material choice of each of the parts; the third defines how the component shrinks in low temperatures; a fourth theory describes whether the component will remain in place also in cold environments; the fifth defines whether the complete component fits in the available space; the final theory expresses the preferences by assigning a cost to the design, based on price, durability, availability, etc.

In order to use these theories to compute a design, we can apply the logical inference task of Model Expansion [11]. 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 predicates such as $Temperature$, $Pressure$ and $Location$ (giving the temperature and pressure ranges and the location in which the component should function); the structure S_{out} then describes a design, by providing an interpretation for predicates such as $SpringShape$ and functions such as $Material$, which maps each component used in the design to the material it should be made from.

However, rather than just computing any model expansion, we make use of IDP's optimisation functionality. This allows us to specify a numerical term t for 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 t is of the form $sum\{p[Penalty] : Violation(p) : p\}$, i.e., we associate to each violation of a preference a certain penalty and the goal is to compute the design for which the sum of all incurred penalties is minimal. IDP implements this inference task by an optimisation loop, which iteratively produces better solutions by each time adding as a new constraint that the next solution must have a lower score than the previous solution. This is the same method as is typically used in, e.g., ASP solvers.

As an implementation of the knowledge base paradigm [5], IDP allows different inference tasks to be performed on the same knowledge base in order to provide different functionalities. Currently, our focus lies on generating designs using the inference task of model expansion. However, in the future, other inference tasks may prove useful for offering additional functionalities, such as explaining why a certain design is not feasible.

6.1 Limitations

Even though using the constraint representation has a lot of interesting advantages, there are also a few downsides to it. The main disadvantage is that it is harder for the domain experts to understand. On the one hand, the syntax for writing down individual constraints is more complex. While we have used IDP because we believe it is quite understandable for untrained experts, it is still much more complex that the simple table-based DMN format. On the other hand, also the constraint-based approach itself seems inherently more difficult for the domain experts. In a DMN decision model, there is always a clear link between input and output, which makes the model easy to interpret and inspect by a domain expert. When using constraints to express design knowledge, a single decision may be affected by numerous constraints. For example in Section 5, the spring design 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.

A second downside is tied to the particular technology used in the IDP system. IDP's model expansion algorithm follows a ground-and-solve strategy (similar to, e.g., ASP solvers), in which all variables are first translated away, by replacing them with all of their possible

values. However, this requires that each variable must have a finite domain, such that the grounding phase can enumerate all of its possible values. Moreover, in order for the grounding to be computed in reasonable time, these domains should be relatively small. Because our application requires some calculations with floating point numbers (e.g., when calculating the shrinkage in cold circumstances), we have had to implement a work-around to perform these calculations outside of the normal ground-and-solve workflow.

7 Validation and Experimental Results

The DMN model. Starting from a direct formalisation of the engineers' design process proved noticeably useful. Not only did the engineers appreciate the intuitive way of reasoning in the DMN standard, it made them think about how they come to a design in a given situation and about why certain design decisions are made. Moreover, when transforming the Visual Basic tool developed for the sales staff into a DMN model, a number of irregularities surfaced. Without a formal representation of this knowledge, it would have been a far more difficult and time consuming task to detect these faults.

To ensure correctness of the DMN model, the engineers not only inspected the decision tables in detail, but also provided us with ten test cases that represent both normal sets of requirements and a number of edge cases. Our OpenRules implementation using the DMN model generates the correct design in all of the test cases. Computing a design takes about 0.3 seconds single core on an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz.

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

The first property to check is that the constraints should not be too strict: for each possible set of inputs I, the design $D(I)$ that would be constructed by the DMN model $D(I)$ should satisfy the constraints in theory T, i.e., $S_I \cup S_{D(I)} \models T$ or in other words, $S_{D(I)}$ is 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 should be minimised. This both checks that the constraints do not fail to rule out designs with a higher score that are in fact impossible and that the weights used in the optimisation criterion are assigned correctly.

We implement both of these checks using IDP. We first transform the DMN model to IDP syntax as described in [4]. We can then use IDP to perform the required checks on 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 to the constraints, the first check was concluded successfully. The second check then revealed that, in a number of cases, the constraint-based model produced more optimal designs than the DMN model. While we initially thought that this was due to more errors in the constraints, an analysis together with the design engineers revealed that the outcome of the constraint-based model was in fact correct and that their own design

process was in these cases non-optimal. This non-optimality turned out to be caused by the difficulty of making the decisions in a fixed order. When using the constraint-based method, no fixed decision order is needed, so a better scoring global optimum can be found.

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.

8 Related Work

A large body of research has been conducted on the topic of automatic product configuration, typically defined as the task of automatically constructing a design from a set of pre-defined components, considering several constraints and some optimisation criteria [2]. Research shows that product configurators have a positive impact on lead time [8, 6] and quotation time [10]. Other comparison studies [18] investigate the effect of configuration systems on product quality, also showing promising results.

A thorough literature review on product configuration was performed by [24]. Their findings reveal that, despite the wide range of existing research, several topics still require further exploration. First, although knowledge acquisition from historical data has been extensively studied, less research has been done on extracting knowledge from domain experts. Moreover, knowledge representation research typically focusses on methods that are intended to be used by knowledge engineers. Little attention has been paid to representations that are usable by domain experts. Our work examines the use of DMN to address these issues in the context of one concrete application domain.

A second aspect which according to [24] 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, [24] also identifies several ways in which additional forms of inference might be useful to provide functionality other than suggesting a design. For example, she identifies such tasks as explaining which conflicting constraints have led to a rejected design or reconfiguring an existing design to cope with changed requirements. The IDP system has been developed according to the knowledge base paradigm [5], in which different logical inference methods can be applied to the same knowledge base in order to implement different functionalities. Both of the tasks of explaining conflicts and of reconfiguration have already been considered in the context of this system [19, 22]. The IDP system therefore provides a suitable formalism to express the design knowledge.

9 Conclusions and future work

In this paper, we have presented an approach to develop a decision support system for the design of mechanical components. This research was conducted in collaboration with a multinational company that wants to standardise and partially automate its design process, both for “routine” applications and challenging new 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 is needed that will allow useful conclusions to be provided to the engineers even in circumstances that fall outside of their designs’ usual scope. On the other hand, the engineers need to be closely involved in the formal specification since they are expected to agree on and understand the model, and to help maintain it.

To cope with these two requirements, we propose a two-step methodology. First, we use the new DMN standard to extract the "routine" design process into an executable formal model, which can already by automatically validated. We then use this DMN model as a basis to perform a second knowledge extraction step, which results in a first-order logic representation that can be given to the state-of-the-art IDP knowledge base system in order to also perform useful inferences in circumstances that fall outside the scope of the routine design process. This IDP model can then be automatically validated w.r.t. the DMN model.

In future work, we plan to examine the possibility of extending the expressivity of DMN to reduce the gap between DMN and IDP, without sacrificing the ease of understanding for the domain experts. Moreover, we also plan to examine the use of IDP's different inference algorithms to address some of the issues highlighting by [24]. Finally, we also wish to develop a method that would allow the more general knowledge expressed in the IDP model to automatically derive DMN design procedures for new application areas.

References

1 Mario Alviano, Francesco Calimeri, Günther Charwat, Minh Dao-Tran, Carmine Dodaro, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Johannes Oetsch, Andreas Pfandler, et al. The fourth answer set programming competition: Preliminary report. In *Logic Programming and Nonmonotonic Reasoning*, pages 42–53. Springer, 2013.

2 David C Brown. Defining configuring. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):301–305, 1998. doi:10.1017/S0890060498124034.

3 M. Bruynooghe, H. Blockeel, B. Bogaerts, B. De Cat, S. De Pooter, J. Jansen, A. Labarre, J. Ramon, M. Denecker, and S. Verwer. Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with IDP3. *Theory and Practice of Logic Programming*, 15(6):783–817, 2015. doi:10.1017/S147106841400009X.

4 Ingmar Dasseville, Laurent Janssens, Gerda Janssens, Jan Vanthienen, and Marc Denecker. Combining DMN and the knowledge base paradigm for flexible decision enactment. In Tara Athan, Adrian Giurca, Rolf Grütter, Marc Proctor, Kia Teymourian, and William Van Woensel, editors, *Supplementary Proceedings of the RuleML 2016 Challenge, RuleML, Stony Brook, 6-9 July 2016*. CEUR-WS.org, 2016. URL: https://lirias.kuleuven.be/handle/123456789/546123.

5 Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Lecture Notes in Computer Science, International Conference on Logic Programming, ICLP, Udine, Italy, 9-13 December 2008*, pages 71–76. Springer, 2008. doi:10.1007/978-3-540-89982-2_12.

6 C. Forza and F. Salvador. Product configuration and inter-firm co-ordination: An innovative solution from a small manufacturing enterprise. *Computers in Industry*, 49(1):37–46, 2002. doi:10.1016/S0166-3615(02)00057-X.

7 Anders Haug, Lars Hvam, and Niels Henrik Mortensen. A layout technique for class diagrams to be used in product configuration projects. *Computers in Industry*, 61(5):409–418, 2010. doi:10.1016/j.compind.2009.10.002.

8 Anders Haug, Lars Hvam, and Niels Henrik Mortensen. 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, 2011. doi:10.1017/S0890060410000636.

9 Lars Hvam, Anders Haug, Niels Henrik Mortensen, Christian Thuesen, Relationship Management, and Product Architecture Group. observed benefits from product configuration systems (Hvam) 2011. *International Journal of Industrial Engineering*, 2013.

10 Lars Hvam, Martin Malis, Benjamin Hansen, and Jesper Riis. Reengineering of the quotation process: application of knowledge based systems. *Business Process Management Journal*, 10(2):200–213, 2004. doi:10.1108/14637150410530262.

11 David Mitchell, Eugenia Ternovska, Faraz Hach, and Raheleh Mohebali. Model expansion as a framework for modelling and solving search problems. Technical report, Technical Report TR 2006-24, School of Computing Science, Simon Fraser University, 2006.

12 N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack. MiniZinc: Towards a standard CP modelling language. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, 2007.

13 Object Modelling Group. Decision Model and Notation, 2016. Version 1.1. URL: http://www.omg.org/spec/DMN/.

14 OpenRules, Inc. OpenRules, 2017. Version 6.4.2. URL: http://openrules.com.

15 F. Salvador and C. Forza. Configuring products to address the customization-responsiveness squeeze: A survey of management issues and opportunities. *International Journal of Production Economics*, 91(3):273–291, 2004. doi:10.1016/j.ijpe.2003.09.003.

16 Fabrizio Salvador, Aravind Chandrasekaran, and Tashfeen Sohail. Product configuration, ambidexterity and firm performance in the context of industrial equipment manufacturing. *Journal of Operations Management*, 32(4):138–153, 2014. doi:10.1016/j.jom.2014.02.001.

17 Alessio Trentin, Elisa Perin, and Cipriano Forza. Overcoming the customization-responsiveness squeeze by using product configurators : Beyond anecdotal evidence. *Computers in Industry*, 62(3):260–268, 2011. doi:10.1016/j.compind.2010.09.002.

18 Alessio Trentin, Elisa Perin, and Cipriano Forza. Product configurator impact on product quality. *International Journal of Production Economics*, 135(2):850–859, 2012. doi:10.1016/j.ijpe.2011.10.023.

19 Hanne Vlaeminck, Joost Vennekens, and Marc Denecker. 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, September 2009. doi:10.1145/1599410.1599428.

20 Bob Wielinga and Guus Schreiber. Configuration-design problem solving. *IEEE Expert-Intelligent Systems and their Applications*, 12(2):49–56, 1997. doi:10.1109/64.585104.

21 Olga Willner, Jonathan Gosling, and Paul Schönsleben. Establishing a maturity model for design automation in sales-delivery processes of ETO products. *Computers in Industry*, 82:57–68, 2016. doi:10.1016/j.compind.2016.05.003.

22 Johan Wittocx, Broes De Cat, and Marc Denecker. Towards computing revised models for FO theories. In Salvador Abreu and Dietmar Seipel, editors, *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*, pages 199–211, November 2009. doi:10.1007/978-3-642-20589-7_6.

23 Dong Yang, Ming Dong, and Rui Miao. Development of a product configuration system with an ontology-based approach. *CAD Computer Aided Design*, 40(8):863–878, 2008. doi:10.1016/j.cad.2008.05.004.

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

25 Linda L. Zhang, Petri T. Helo, Arun Kumar, and Xiao You. Implications of product configurator applications: An empirical study. *IEEE International Conference on Industrial Engineering and Engineering Management*, 2016-Janua(i):57–61, 2016. doi:10.1109/IEEM.2015.7385608.