# 20th International Configuration Workshop

## Proceedings of the 20th International Configuration Workshop

*Edited by*
*Alexander Felfernig, Juha Tiihonen,*
*Lothar Hotz, and Martin Stettinger*

September 27 – 28, 2018

Graz, Austria

Organized by



University of Hamburg
Hamburger Informatik Technologie-Center e.V.
Department of Computer Science
Vogt-Kölln-Str. 30, 22527 Hamburg
GERMANY

ISSN 1613-0073

Alexander FELFERNIG, Juha TIIHONEN, Lothar HOTZ, and Martin STETTINGER, Editors
Proceedings of the 20th International Configuration Workshop
September 27-28, 2018, Graz, Austria

## Chairs

*Alexander Felfernig*, Graz University of Technology, Austria
*Juha Tiihonen*, University of Helsinki, Finland
*Lothar Hotz*, University of Hamburg, HITeC, Germany
*Martin Stettinger*, Graz University of Technology, Austria

## Program Committee

*Michel Aldanondo*, Toulouse University, Mines Albi, France
*Tomas Axling*, Tacton Systems, Denmark
*Andrés Felipe Barco*, Universidad de San Buenaventura-Cali, Colombia
*David Benavides*, University of Seville, Spain
*Andreas Falkner*, Siemens AG, Austria
*Alexander Felfernig*, Graz University of Technology, Austria
*Cipriano Forza*, University of Padova, Italy
*Gerhard Friedrich*, University of Klagenfurt, Austria
*Paul Grünbacher*, Johannes Kepler University Linz, Austria
*Albert Haag*, Product Management GmbH, Germany
*Alois Haselböck*, Siemens AG, Austria
*Petri Helo*, University of Vaasa, Finland
*Lothar Hotz*, University of Hamburg, HITeC, Germany
*Dietmar Jannach*, University of Klagenfurt, Austria
*Katrin Kristjansdottir*, Technical University of Denmark, Denmark
*Yiliu Liu*, Norwegian University of Science and Technology, Norway
*Tomi Männistö*, University of Helsinki, Finland
*Mikko Raatikainen*, Aalto University, Finland
*Rick Rabiser*, Johannes Kepler University Linz, Austria
*Sara Shafiee*, Technical University of Denmark, Denmark
*Markus Stumptner*, University of South Australia, Australia
*Juha Tiihonen*, University of Helsinki, Finland
*Elise Vareilles*, Toulouse University, Mines Albi, France
*Yue Wang*, Hang Seng Management College, Hong Kong
*Linda Zhang*, IESEG Business School of Management Paris, France

## Local Arrangements

*Alexander Felfernig*, Graz University of Technology, Austria
*Elisabeth Orthofer*, Graz University of Technology, Austria
*Martin Stettinger*, Graz University of Technology, Austria

Sponsors


SIEMENS
Ingenuity for life


encoway
standard for variety


PMH


variantum


GRAZ


TU
Graz


AGILE


OpenReq


selection:arts
Intelligent Decision Technologies


EventHelpr

# Preface

Configuration is the task of composing product models of complex systems from parameterisable components. This task demands for powerful knowledge-representation formalisms to capture the great variety and complexity of configurable product models. Furthermore, efficient reasoning and conflict resolution methods and are required to provide intelligent interactive behavior in configurator software, such as solution search, satisfaction of user preferences, personalization, or optimization.

The main goal of the Configuration Workshop is to promote high-quality research in all technical and application areas related to configuration. In this year, besides typical contributions about knowledge representation and reasoning in configuration, interacting with configurators is a main focus.

The workshop is of interest for both, researchers working in the various fields of Artificial Intelligence (AI) technologies as well as industry representatives interested in the relationship between configuration technology and the business problem behind configuration and mass customization. It provides a forum for the exchange of ideas, evaluations and experiences especially in the use of AI techniques within these application and research areas.

The 2018 Workshop on Configuration continues the series of workshops started at the AAAI'96 Fall Symposium and continued on IJCAI, AAAI, and ECAI since 1999. In recent years, the workshop was held independently from major conferences.

Alexander Felfernig, Juha Tiihonen, Lothar Hotz, and Martin Stettinger

August 2018

# Contents

# Product Configuration in the Wild: Strategies for Conflicting Decisions in Web Configurators

**Thomas Thüm**[1] and **Sebastian Krieter**[2] and **Ina Schaefer**[1]

**Abstract.** Customization is omnipresent in our everyday live. There are web configurators to customize cars, trucks, bikes, computers, clothes, furniture, and food. At first glance, customization using configurators appears trivial; we simply select the configuration options that we want. However, in practice, options are usually dependent on each other. Reasons for dependencies are manifold and are typically specific for the particular domain. Dependencies can be simple, such as one option requiring or excluding another option, but also arbitrarily complex, involving numerous options. In this study, we aim to understand how today's web configurators support users in their decision making process. In particular, we are interested in understanding how configurators handle decisions that are in conflict with dependencies. To abstract from different visualizations, we classify the existing strategies of web configurators and discuss advantages and disadvantages of them. While we identified eight strategies, a single configurator typically uses several of those strategies.

## 1 Introduction

Mass customization is the vision that customized products are produced to a price similar to that with mass production [3, 4, 13, 15]. This vision requires that customers specify their needs explicitly [10]. To this end, there are thousands of product configurators available in the world wide web,[3] which guide customers during the decision making process. Unfortunately, customers do not only need to understand which options are available, but also which combinations of options are valid. For the success of mass customization it is crucial that potential customers are able to easily explore valid combinations and to make compliant decisions. Our personal experience with product configurators suggests that we are not there yet, as configurators often require a considerable mental effort from their users.

In Figure 1, we show an excerpt of a configurator for a Lenovo ThinkPad. The excerpt contains three configuration options, of which each stands for a different display being available for the notebook. However, these three options cannot be chosen freely due to existing dependencies. First, the options are alternatives to each other, meaning that a notebook must have exactly one of those three displays. For that purpose, the three options are arranged in a category called *Display*. Second, the green hint below the last option reveals that this display is only available if option *WWAN* (not shown in the excerpt) is chosen. While the first dependency is enforced by the configurator, the second must be taken care of by the user. We argue that dependencies are one of the main challenges for customers, as they heavily influence the decision making process.

[1] TU Braunschweig, Germany
[2] University of Magdeburg, Germany
[3] `https://www.configurator-database.com/`

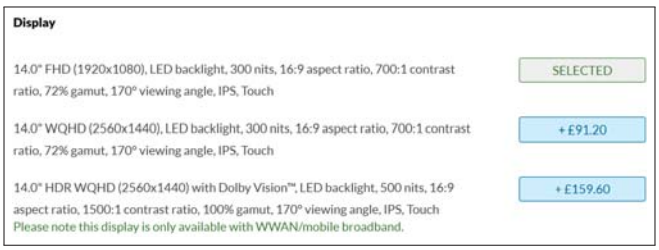


**Figure 1.** Example dependency in Lenovo's ThinkPad configurator.

While there are several approaches to implement configurators [5, 15], we aim to understand how configurators handle dependencies from a user's point of view. That is, we want to identify strategies to guide the decision process, which are currently applied in real-world configurators. Our insights may be used to uncover gaps between research and practice on product configuration. In particular, we analyze advantages and disadvantages of each strategy, which may support development of more sophisticated strategies in the future.

By studying a corpus of seven web configurators (cf. Section 2), we identified eight strategies to handle dependencies and discuss their advantages and disadvantages (cf. Section 3). We focus on web configurators, because they are freely available and are used by thousands or even millions of customers. Interestingly, of each configurator uses a different subset of all identified strategies. Moreover, configurators with fewer categories seem to be simpler in a sense that they apply fewer strategies. In particular, we even studied a configurator applying all eight strategies.

## 2 Subject Configurators

As considering all product configurators on the web is certainly not feasible, finding a representative selection of configurators is crucial for our study. Initially, we googled the term "configurator" and inspected the first 100 search results to understand which configurators are used frequently on the web. Almost all entries that we found were configurators for cars, but we wanted to cover more than just the automotive industry. We though of further industries where we experienced mass customization before. Then, we tried to find representative configurators for those industries that are likely to be used by a wide audience. Ultimately, we have chosen configurators from the domain of automotive, mobile computers, smartphone accessories, and clothing.

In our study, we do not distinguish between configurators that are used for customized production opposed to a selection of pre-produced goods (aka. selectors), as this distinction is typically not visible to customers anyway. Even though the number of potential

combinations and length of the delivery period might be good indicators. For instance, in order to speed up delivery, t-shirts are typically available in different sizes and colors, which are often produced before customers make their decision.

As we aim to provide evidence for our study by means of screenshots, we decided to use the UK version of all configurators. Nevertheless, we have drawn samples in other languages, too. The only differences we identified were options being available only in some countries, whereas the principle strategies have been identical. We accessed all configurators in May 2018, whereupon we experienced that configurators and their behaviors change on a daily basis. That is, dependencies are updated frequently, whereas we did only experience changes in the strategy for one configurator (cf. Section 3.2).

**BMW 1 (3 Door)** BMW is a German car manufacturer providing individual configurators for 49 different car models.[4] As cars are known to have hundreds of configuration options, we decided to use the configurator for the cheapest car model to find a comparatively small, and thus manageable number of configuration options for our study. There are 42 alternative options for engines and gearboxes, eleven exterior colors, five alloy wheels, three options for upholstery, four interior designs, two packages, and 81 further options with respect to optional equipment.

**Toyota AYGO x-play 5 Door Hatchback** With about 9 million sold cars a year, Toyota is one of the largest car manufacturers. Toyota offers 19 models, which are available in a total of 84 editions of which each can be configured separately. Just as for BMW, we wanted to select the cheapest model and edition available, namely Toyota AYGO x3 Door Hatchback. However, users cannot even choose anything in the first configuration step for this edition. Thus, we have chosen the cheapest edition for this model for which users can choose between manual and automatic gearshift, namely Toyota AYGO x-play 5 Door Hatchback.[5]

**HP Velotechnik Streetmachine GTE** Besides cars, many other transportation means exist that can be configured. In particular, bicycles can often be configured by customers. We investigated the configurator of HP Velotechnik, as we already had experience with this configurator prior to this study. Customers can choose one of 14 recumbent bicycles and then start the configurator. We have chosen the model Streetmachine GTE which provides 34 categories of which customers have to chose one option each.[6]

**Lenovo ThinkPad X1 Yoga (3rd Gen)** ThinkPad X1 is one of the most expensive and powerful convertibles of Lenovo. Before users can configure it on Lenovo's website,[7] they have to choose between three models in black and one model in silver. Of those, we have chosen the cheapest as it seems to provide the most configuration options (i.e., many options could not be downgraded in the more expensive models). The configurator provides 39 configuration options in eleven categories, such as processor, operating system, display, hard drive, and keyboard. Besides those, there are another eleven categories with a single option (i.e., cannot be selected by the user).

**Microsoft Surface Book 2** Surface Book 2 is the latest high-end convertible by Microsoft. With 13 configuration options in five categories there is only some rudimentary support for customization. On Microsoft's website there is a link to pre-configured products,[8] but also to different versions of a configurator to which we later refer to as a version with a default configuration[9] and a version without a default configuration.[10]

**T-Shirts at Amazon** Online shopping is an increasing market and Amazon is one of the most popular web stores. Although not all products can be configured, there is huge amount of products for which at least certain properties, such as color and size, can be specified. Even though there are many products, the configurator technology seems to be rather generic at Amazon. For our study, we have chosen a t-shirt configurator, as t-shirts are a rather common consumer good and typically have two categories, which is necessary to study dependencies. The subject configurator offers 21 colors and 8 sizes, whereas 123 of 168 combinations are valid (i.e., 73.2%).[11]

**Smartphone Cases at Ebay** Another popular web store is Ebay, which appears to be similar in terms of configuration compared to Amazon. Again, the underlying configurator seems to have the same behavior for very different kinds of products. Nevertheless, we have to decide on one product for our case study, for which we have chosen a configurator for smartphone cases.[12] The configurator supports the selection out of 36 smartphone models and eight colors. Overall, 208 out of 288 principal combinations are valid (i.e., 72.2%).

## 3 Strategies for Conflicting Decisions

We investigate all seven configurators to explore how they deal with conflicting customer decisions. For that purpose, we randomly selected options and observed whether those decisions had any consequences for the selection of other options. We identified the following eight strategies, whereas we abstract from the visual representation of selected and deselected options.

### 3.1 Automatic Deselection in Alternatives

A simple but effective strategy exists for alternative options within one category. If an option A is already selected and a user attempts to select an alternative option B in the same category, a configurator may automatically deselect option A. Users are typically not notified about this change.

All configurators that we studied apply this strategy. As an example, we refer again to Figure 1 and the display options of a ThinkPad. If a user selects the second or third display option, the first display option is deselected (i.e., the price difference is shown).

The automatic deselection avoids further burden on the user for these kinds of simple conflicts, as the conflict is immediately solved. It is also an intuitive strategy, especially if accompanied with a graphical representation that suggests that these options are alternative to

[4] https://www.bmw.co.uk/configurator#/
[5] https://www.toyota.co.uk/new-cars/aygo/build
[6] http://hpvelotechnik.velocom.de/step-1.jsf
[7] https://www3.lenovo.com/gb/en/laptops/thinkpad/x-series/ThinkPad-X1-Yoga-3rd-Gen/p/22TP2TXX13Y
[8] https://www.microsoft.com/en-gb/surface/devices/surface-book-2/
[9] https://www.microsoft.com/en-gb/store/config/surface-book-2/8MCPZJJCC98C/17NG
[10] https://www.microsoft.com/en-gb/store/config/surface-book-2/8MCPZJJCC98C
[11] https://www.amazon.co.uk/gp/product/B00V3IDB3Q
[12] https://www.ebay.co.uk/itm/Sports-Running-Gym-Cycling-Jogging-Armband-Case-Cover-For-Huawei-Mobile-Phones/181809023183

each other (e.g., a drop down box as in Figure 3 or radio buttons as in Figure 5). Nevertheless, the deselection is problematically if the user does not recognize it (e.g., if the deselected option is not on the screen). Furthermore, the deselection itself may result in further conflicts with other decisions.

## 3.2 Starting with a Default Configuration

Dependencies are typically unsatisfied when no options are selected at all. For example, a notebook needs exactly one kind of display. A very common strategy to avoid invalid states when starting the configuration process is to start with a default configuration. The default configuration satisfies all dependencies and may be changed by the user during the process.

For instance, the first display is already selected when opening Lenovo's configurator (cf. Figure 1) and in BMW's configurator a default engine is selected from the start. Interestingly, Microsoft's configurator can be accessed via different links of which some start with a default configuration and others do not. However, even when starting with a valid default configuration in Microsoft's configurator, any change resets decisions on later categories. As a consequence, changing the display to 15 inch removes all default selections and makes the configuration invalid (cf. Figure 2). It requires decisions on all subsequent categories to finish and to come to a valid configuration again. Note that all decisions of subsequent categories are reseted independent of whether they are actually conflicting or not.[13]

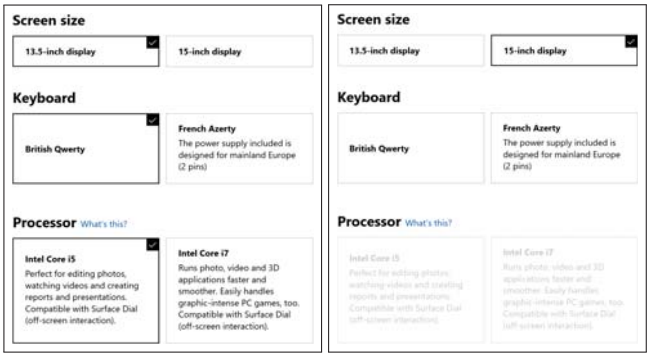


**Figure 2.** Default configuration and its reset in Microsoft's configurator.

Starting with a valid configuration means that a user can also skip the configuration process completely. It may also help users to avoid making some decisions, if they do not care about all options. They simply have to change options they care about. However, the downside of starting with a default configuration is that users can often not recognize which options they have chosen and which have already been chosen by the default configuration. As a consequence, they may forget to examine some of the decisions and may end up with a product not customized to their needs.

## 3.3 Hiding Invalid Combinations

Another simple strategy to deal with conflicts is to hide all options that are in conflict with previous decisions. At the start, all options of all categories are visible and no option is selected by default. Once a user makes a selection in one of the categories, all other categories are filtered for compatible options. As a result, no decision of the user leads to a conflicting state.

This strategy is followed by the configurator available to products at the Ebay web store. All models and all colors are shown when nothing is selected. For a given color, between 22 and 33 models are visible, and for a given model, there are between one and all eight colors. In Figure 3, we selected the *Huawei Ascend P7*, for which a case is only available in black, blue, and gray.

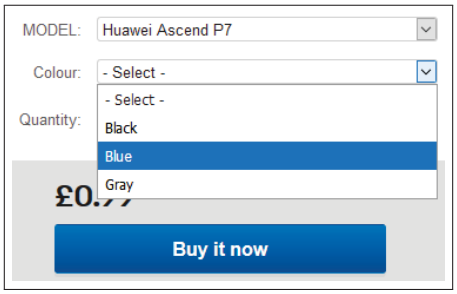


**Figure 3.** Hiding of invalid combinations in the Ebay configurator.

The advantage of hiding invalid combinations is that users are not overwhelmed by a large number of options that are actually not available for their prior decisions. Also, there is no need to solve any conflicts and users only need to decide on the remaining valid options. Nevertheless, there are also major drawbacks. This strategy seems to be rather incompatible with a default configuration and, thus, it will not be possible to simply continue and skip decisions. Requiring users to make every decision is fine if there are only a few categories, but can be infeasible for large configuration spaces. Nevertheless, we found such a combination in the configurator by Toyota: a default car color is selected when entering the configurator for which black wheels are filtered out. Hence, users will only find out about black wheels if they change the car color first. Furthermore, this strategy might be unintuitive to users who do not expect such a filtering. Indeed, it took us a while to find out that the lists in the Ebay configurator are changed depending on other selections. Finally, it seems that users should start with selecting the most relevant criteria for them, because otherwise they will simply not notice that there are other, potentially better options for later selected categories.

## 3.4 Alternatives of Compound Options

Most product configurators have categories with alternative options (i.e., options of which exactly one or at most one option can be chosen). If there are dependencies between these categories, one strategy is to build pairs of the corresponding options that are wanted. Those alternative pairs are then grouped under one compound category. Pairs not allowed by dependencies are simply omitted.

For instance, in the ThinkPad configurator, dependencies between the categories *Microsoft Office* and *Adobe Acrobat* are handled this way. Office is available as *365 Home*, *365 Personal*, *Home and Student*, *Home and Business*, and *Professional*. However, what is available does also depend on whether *Acrobat* is selected: *Professional* requires *Acrobat* and *Acrobat* requires *Home and Business* or *Professional*. Instead of presenting six options for *Office* and two options for *Acrobat* with two dependencies, there are seven compound options without dependencies, as illustrated in Figure 4.

[13] When we tested the configurator in April 2018, the configuration in subsequent categories was even re-assigned automatically. This behavior was even worse from a user's point of view as decisions on all subsequent categories are not just requested again, but even overridden without notice. It is likely that this behavior was replaced as being too unintuitive or even questionable from a legal perspective.

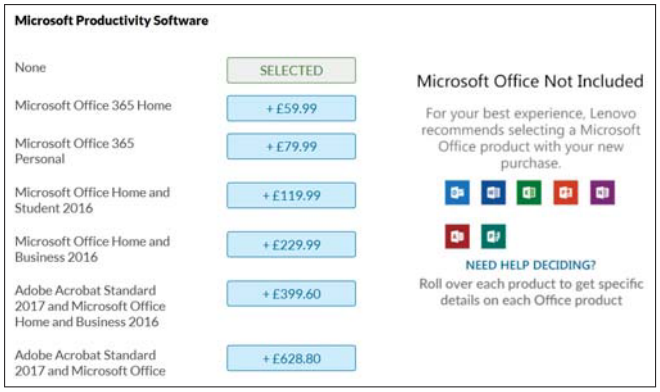


**Figure 4.** Compound options in Lenovo's ThinkPad configurator.

BMW uses compound options for dependencies between 23 engines and three gearbox options. Out of 69 combinatorial combinations, only 42 are considered valid. In particular, four engines are not available with manual gearbox and engines are either available with *Automatic Gearbox* or with *Sport Automatic Transmission* (i.e., $4 + (23 - 4) * 2 = 42$). A possible reason for compound options besides eliminating constraints is that the choice of engine and gearbox affects the acceleration, consumption, $CO_2$ emission, and pricing, which are properties shown next to each option (cf. Figure 5).


**Figure 5.** Compound options with non-functional properties for a BMW.

Compound options can essentially be used to get rid of all dependencies, which results in an enumeration of all products. Microsoft gives customers both options, a configurator and a simple selection among pre-configured products as shown in Figure 6. Interestingly, our comparison of available products revealed that the keyboard is always pre-configured to be *British Qwerty* and else only one products is missing opposed to the product configurator (i.e., *Intel Core i5* with *128GB* storage). That is, ten configuration options in four categories have so many constraints that they only result in eight different products for customers, instead of 32 theoretical combinations.

Compound options are well-suited if the combined options have many dependencies and customers can directly explore the few available combinations. In particular, it would not be feasible to present all 123 combinations of colors and sizes in the Amazon configurator (cf. Figure 10). However, compound options also typically lead to the problem that one original option is represented by several compound options, which increases the effort for customers when comparing

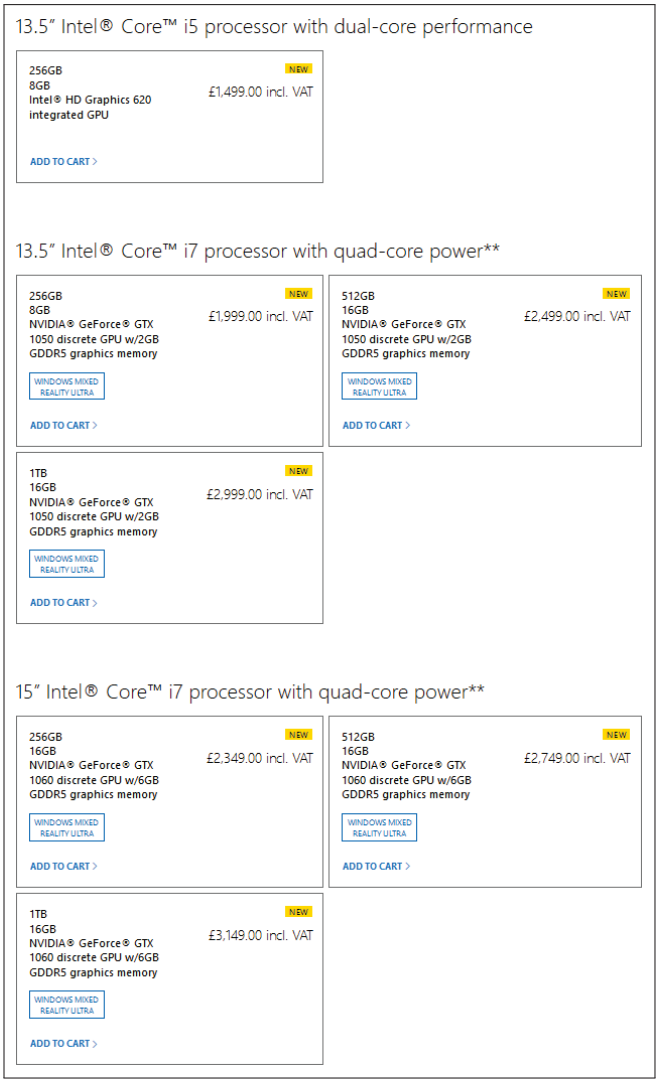


**Figure 6.** Microsoft's product selection opposed to configuration.

compound options. Furthermore, compound options may complicate the handling of dependencies to options in other parts of the configurator. For instance, a simple rule requiring another option may then need to be duplicated for all occurrences in compound options.

## 3.5 Continuing with Invalid States

A further strategy to deal with conflicting options is to let users continue with the configuration process although the current configuration is invalid. That is, users may even select an option that is not allowed according to dependencies and already selected options. The action required by the user to fix the configuration is basically postponed to a later point during the configuration process.

As illustrated in Figure 7, Lenovo's configurators issue a warning to the user once a conflict was detected. The user can continue with the configuration process, but the same warning appears after every selection until the problem is fixed. Unfortunately, the only hint on how to solve this problem is the written-down dependency along with one of the interacting configuration options. Thus, the dependency may happen to be written next to the option whose selection lead to the problem or next to any other option.

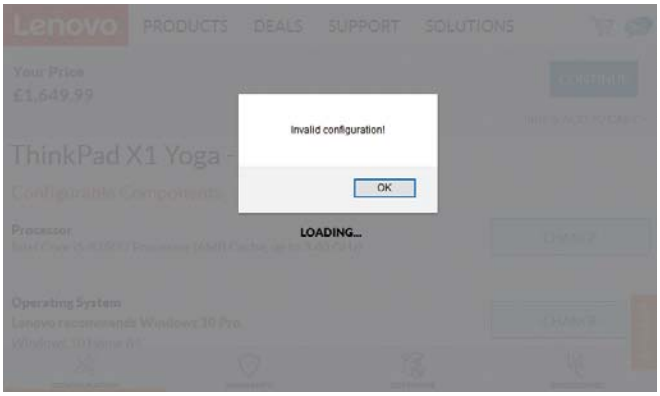


**Figure 7.** Warning for invalid states in Lenovo's ThinkPad configurator.

Letting users continue with invalid states seems to be a flexible way for configuration, as users can decide when to handle conflicts. In particular, a user can focus on the most relevant options first and then solve conflicts with the remaining options. However, especially problematic is that postponing a fix can result in multiple conflicts, which are even harder to fix. Furthermore, it seems that tool support for invalid states is not straightforward, as we have only found this strategy in the two configurators by Lenovo and Toyota, of which neither provided useful support.

## 3.6 Subsequent Configuration Steps

Configurators with many options often split the decision process over several steps. The decisions in one configuration step are confirmed before continuing with subsequent steps. While configurators typically allow to go back and forth in those steps, the configuration decisions of subsequent configuration steps may be reverted in this process. In some configurators, certain steps can even be skipped based on the configuration in prior steps.

For instance, the BMW configurator consists of seven steps and depending on the choice of engine, a further additional step called *Model Variants* may appear (cf. Figure 8). However, in the studied configurator, the additional step was only used to show an inclusive option called *Sport package*, which cannot be deselected.

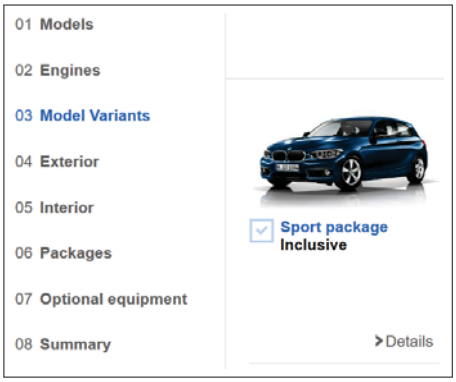


**Figure 8.** Configuration steps in the BMW configurator.

Configuration steps seem to increase the overview over complex configuration processes with hundreds of features. Hiding irrelevant configuration steps is also an interesting concept to deal with constraints. Nevertheless, hiding certain steps may also be a challenge for users that are actually looking for a given step and do not understand why it is not visible. In addition, users could be unaware of configuration steps that would be helpful in their decision making process. Furthermore, although guiding the user and hiding options, configuration steps do not solving the problem of conflicting configuration options. Even worse, constraints between different configuration steps could be a particular challenge for users, if they have to switch back and forth over and over again.

## 3.7 Automated Reconfiguration

Another strategy to deal with constraints is to automatically select and deselect options after every user decision. In particular, constraints are used to identify conflicting decisions. Those conflicts are then resolved automatically, if possible.

Automated reconfiguration is applied by the BMW configurator. If already chosen configuration options are affected by the current decision, a configuration assistant is shown to the user summarizing the automated changes (cf. Figure 9). There is also an automatic mode in which the configuration assistant will not inform users after every reconfiguration. Unfortunately, we were not able to provoke a situation in which the configurator obviously has to chose one of several alternative fixes. Such a situation would be interesting to see whether some kind of user interaction is supported in this process.

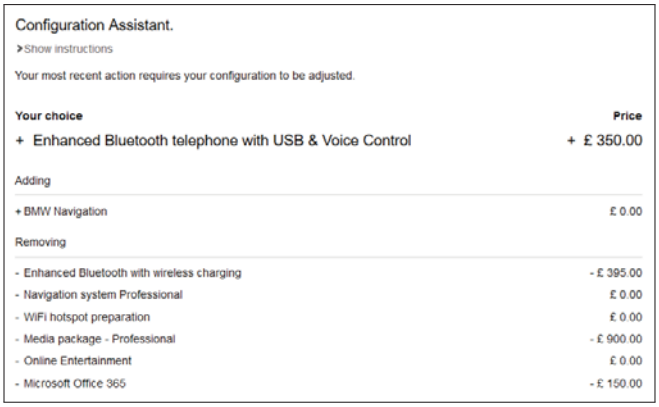


**Figure 9.** Automated reconfiguration in the BMW configurator.

In the Amazon configurator, we can distinguish available from unavailable combinations visually, as illustrated in Figure 10. Nevertheless, we can select also grayed-out options, such that a conflicting state arises. How the conflict is solved dependent on the latest decision. If a new size is selected that is not available for the current color selection, then another color that is still available is automatically selected. The selection appeared to be arbitrary, but it could be based on a recommender knowing more common combinations (e.g., red for smaller sizes and blue for larger ones). Interestingly, if a new color is selected the configurator does not automatically select another size, but instead resets the size selection. A reason for this different strategy might be that people can come to terms with another color, but typically are not willing to choose a t-shirt in a great color with a size that does not fit them.

One advantage of automated reconfiguration is that conflicts are directly solved when they occur. Furthermore, users may have the chance to confirm or abort the current decision propagation. However, an issue with this strategy is that reverting decisions that triggered automated reconfiguration may not revert the automatically introduced changes. This behavior can be confusing to the user and can

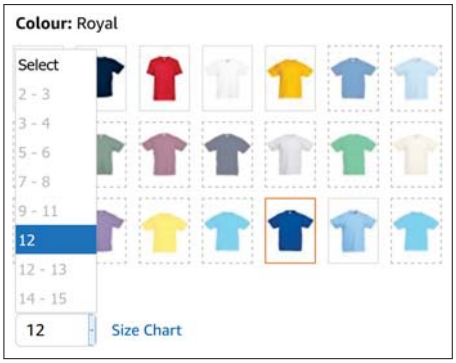


**Figure 10.** Automated reconfiguration in the Amazon configurator.

even introduce further conflicts. Furthermore, automated reconfiguration can interact with other applied strategies in undesired ways, which can again lead to the introduction of conflicts.

## 3.8 Interactive Resolution of Conflicts

Once a conflict occurred, it can either be resolved automatically, as discussed in Section 3.7, or with user interaction. With interactive resolution, we refer to a process in which the configurator informs users about conflicts in the current configuration and guides them to a resolution by proposing possible changes. Thus, interactive resolution is especially helpful if there are several meaningful resolutions to a conflict and the users input is required.

We experienced interactive resolution of conflicts in the configurators by Toyota and by HP Velotechnik. For Toyota, we identified a dependency between car colors and wheels. Black wheels are only available for three of six car colors, namely *White Flash*, *Electro Gray*, and *Bold Black*. Only when of those colors is selected, black wheels are visible (cf. hiding invalid combinations). However, when black wheels are selected, we can still choose one of the three incompatible car colors, as those are not hidden. In that case, the configurator selects the incompatible car color and shows the dialog for wheels in which the previously chosen wheel is hidden. If users select another wheel, the conflict is resolved. However, we experienced

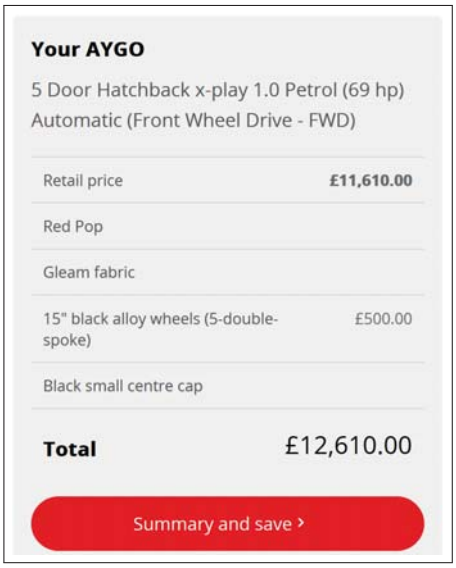


**Figure 11.** An unresolved conflict leading to a wrong price computation in the Toyota configurator.

bugs when users decide to cancel the interactive process. We were actually able to configure a car with a wrong price computation (cf. Figure 11). If we continue and try to actually buy that car, the reason for the wrong computation becomes clear; the configurator automatically selected two wheel types, for which eight instead of four wheels are charged in terms of costs (cf. Figure 12). When we tried to fix the conflict manually, we were not even able to open the configuration dialog for wheels anymore. However, the Toyota configurator has a rather unique feature allowing users to undo and even redo previous decisions by going back and forth in the browser.


**Figure 12.** Two alternative wheels selected in the Toyota configurator.

For HP Velotechnik, interactive resolution is applied to all conflicts that occur between any pair of two categories. When opening the configurator, each category is set to the first option, whereas options are sorted in alphabetical order. When changing the tires to *Schwalbe Kojak* for the default configuration, a dialog opens asking the user to select one of two options as a drivetrain (cf. Figure 13). Although we opened the UK version, there is a hint in German saying that an unrelated drivetrain (i.e., not available and not selected as default) has some conflicts with particular brakes. After studying the configurator for a while, we found out that this hint is always shown if there is a conflict with drivetrains and the hint is most likely supposed to help with another conflict. Selecting one of those two options leads to another conflict which is resolved with the same technique. While, in this case, there are only two interactive resolution steps due to one selection, we experienced cascades of up-to five conflicts. With such cascades, there is a considerable burden on the user to understand why all those changes need be done. Configuration is especially challenging as users easily get lost which options were the default, which did they chose on purpose, and which options had to be selected in the interactive conflict resolution.

The advantage of resolving conflicts interactively is that the user is essentially making decisions on conflicting options. Hence, opposed to automatic reconfiguration users have the control over conflict resolution and opposed to the hiding of invalid combinations they can actually see and decide on all options. The downside is, however, that users have to take decisions in response to a conflict directly. In particular, this may lead to a frustrating cascade of numerous conflict resolutions, which may even need to be reverted, if the user is not satisfied with the options in one conflict resolutions. A postponed interactive resolution would probably need some kind of support for continuing with invalid states (cf. Section 3.5), which seems to be hard to be implemented properly.

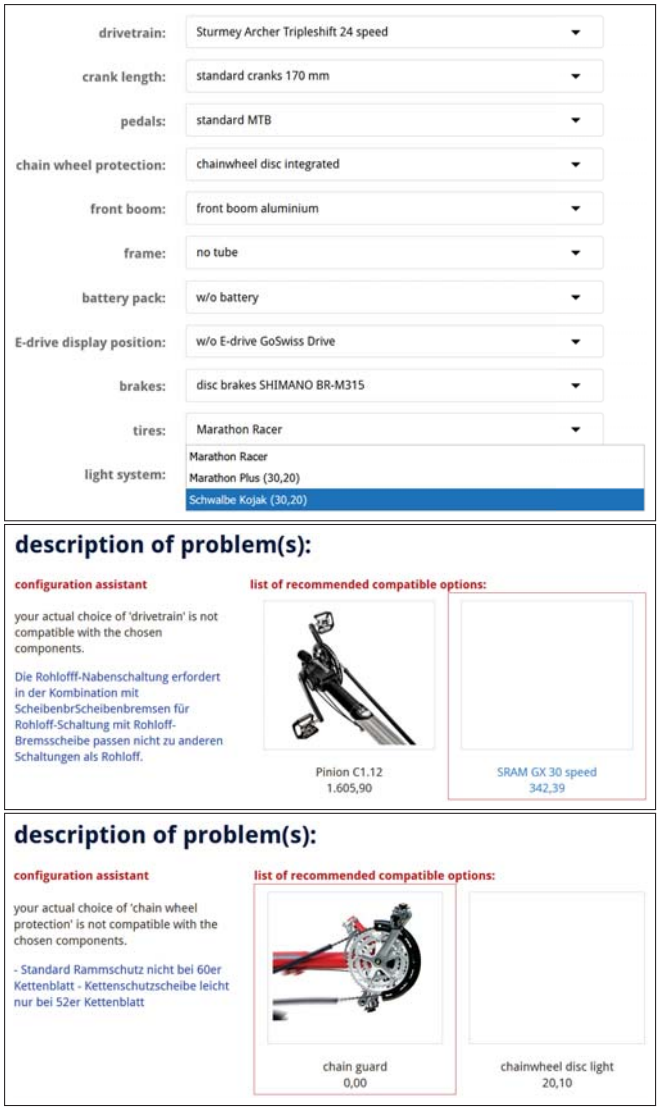


**Figure 13.** A cascade of conflicts in the HP Velotechnik configurator.

## 3.9 Combinations of Strategies

In the previous sections, we discussed each identified strategy independently. However, in practice, none of the studied configurators applies only a single strategy. Instead, we found that configurators apply between two and eight strategies, as illustrated in Table 1. HP Velotechnik, Ebay, and Amazon only apply two or three strategies. The configurators of Ebay and Amazon did only come with a few categories of alternative options. The most strategies are applied in the car configurators by Toyota and BMW, which also comprise the largest number of options and categories. The configurator by Toyota even applies all eight strategies.

We can further distinguish between strategies that are applied globally to all options or only locally to some options. Automatic deselection of alternative options is not only applied in all configurators, but also for all categories with alternative options. Similarly, configuration steps and default options have been applied only globally. In contrast, all other strategies have typically be applied locally to some options in the configurator. One exception is that the Ebay configurator uses hiding of invalid combinations in all parts, but there were only two categories of alternative features. Other exceptions

| Strategy | BMW | Toyota | HP Velotechnik | Lenovo | Microsoft | Amazon | Ebay |
|---|---|---|---|---|---|---|---|
| Automatic Deselection in Alternatives | y | y | y | y | y | y | y |
| Default Configuration | y | y | y | y | y/n | y | n |
| Hiding Invalid Combinations | n | y | n | n | n | n | y |
| Compound Options | y | y | n | y | y | n | n |
| Continuing with Invalid States | n | y | n | y | n | n | n |
| Subsequent Configuration Steps | y | y | n | n | n | n | n |
| Automated Reconfiguration | y | y | n | n | y/n | y | n |
| Interactive Resolution of Conflicts | n | y | y | n | n | n | n |

**Table 1.** Overview on configurators and their strategies.

are that BMW applies automated reconfiguration globally and HP Velotechnik uses interactive resolution for all categories.

The configurator by Microsoft is especially interesting as there are multiple links to configurators for the same product, which apply different strategies. For instance, there is a version of a configurator using default configurations and another version that opens without any decisions. Also, we experienced that the Microsoft configurator stopped using automated reconfiguration in April or May 2018.

The strategies that we identified can be further classified into strategies *avoiding* all or at least some conflicts and those strategies *resolving* conflicts. Default configurations, hiding invalid combinations, compound options, and configuration steps help to avoid conflicts. All other strategies let users make conflicting decisions and later solve them automatically or interactively.

## 4 Related Work

There exists a number of previous studies comparing web-based configurators [1, 14, 12]. Each of these studies considers different aspects of these tools. For instance, Streichsbier et al. focuses on the differing user interface of configurators for different domains [14]. They describe differences and similarities in the design of configurators, such as navigation buttons, product images, and prices displays, across multiple domains. Abbasi et al. compare 111 different configuration tools regarding the process of configuration and the internal behavior of configurators [1]. In their work, they consider the visual representation of options and constraints and the corresponding semantics behind it, constraint syntax and handling, and control flow of the configuration process. Pil and Holweg studied configurators of ten car manufacturers [12]. In addition to the external variety provided by the configurator, they also studied the internal variety by surveying the manufacturers. In contrast to these prior studies, we explicitly focus on the underlying strategies for handling conflicts and categorize these further.

Other work, such as Herrmann et al., Haag et al., and Hubaux et al. have looked at specific strategies that we cover in our paper [6, 7, 9]. In their paper, Herrmann et al. investigate the effects of starting the configuration process with a *default configuration*, which can be adapted by the user [7]. Although we do not investigate user behavior when using a particular conflict handling strategies, we aim to provide a broader view of currently used strategies on the web. Haag et al. consider different techniques for finding explanations in conflicting configurations [6]. This is corresponds to our identified strategy of *resolving conflicts interactively*. Amongst others, Hubaux et al. investigate which conflict handling strategies are being used in the two product lines Linux and eCos [9]. Although they are

more focused on the user perspective, they find that the Linux' configuration tool uses a mix of different strategies, such as *automated reconfiguration*, *hiding invalid combinations*, and *resolving conflicts interactively* and that eCos also allows to *continue with invalid states*.

Product configuration is very similar to software configuration [8], for which many tools exist [2, 11]. Berger et al. conducted a survey among practitioners and researchers with industrial experience on software product-line engineering and variability modeling [2]. Amongst others, they present data about the distribution of modeling and configuration tools that are used in practice and the scale of real-world variability models, regarding number of configuration options and dependencies. While, in this paper, we are also interested in dependencies that result from variability modeling, we foremost consider the strategies to enforce these dependencies during the configuration process.

## 5 Conclusion and Future Work

Product configurators are essential for the vision of mass customization. Customers use configurators to identify options and their valid combinations. A configurator is often the main source of knowledge being queried in the decision making process. We studied a corpus of seven configurators to understand how they handle dependencies between configuration options. Those dependencies can easily lead to decisions by a customer that are in conflict with each other. We argue that how a configurator supports users in avoiding or fixing conflicts is crucial for the success of mass customization.

In our study, we identified eight strategies to handle dependencies. Half of those strategies aim to prevent conflicting decisions whereas the rest help customers to resolve conflicts automatically or with their interaction. Most strategies are applied by several but not all configurators, whereas one strategy is applied by all configurators and all other strategies are applied by at least two configurators. Similarly, each studied configurator applies at least two strategies and we also found a configurator that applies all eight strategies. It seems that smaller configurators with fewer options and fewer dependencies tend to use fewer strategies and larger configurators apply numerous strategies.

While we discussed advantages and disadvantages of all strategies, it is an open research question which combinations of those strategies are best in which situations. Furthermore, it would be interesting to investigate whether there are further strategies and how each strategy can be supported with the state-of-the-art techniques for the implementation of configurators. In any case, studying further real-world configurators may give more insights.

## REFERENCES


[1] Ebrahim Khalil Abbasi, Arnaud Hubaux, Mathieu Acher, Quentin Boucher, and Patrick Heymans, 'The Anatomy of a Sales Configurator: An Empirical Study of 111 Cases', in *Proc. Int'l Conf. Advanced Information Systems Engineering (CAiSE)*, pp. 162–177, Berlin, Heidelberg, (2013). Springer.

[2] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski, 'A Survey of Variability Modeling in Industrial Practice', in *Proc. Int'l Workshop Variability Modelling of Software-Intensive Systems (VaMoS)*, pp. 7:1–7:8, New York, NY, USA, (2013). ACM.

[3] Stanley M. Davis, *Future Perfect*, 1987.

[4] Rebecca Duray, Peter T. Ward, Glenn W. Milligan, and William L. Berry, 'Approaches to Mass Customization: Configurations and Empirical Validation', *J. Operations Management (JOM)*, **18**(6), 605–625, (2000).

[5] Cipriano Forza and Fabrizio Salvador, *Product Information Management for Mass Customization: Connecting Customer, Front-Office and Back-Office for Fast and Efficient Customization*, Palgrave Macmillan, New York, NY, USA, 2006.

[6] Albert Haag, Ulrich Junker, and Barry O'Sullivan, 'A survey of explanation techniques for configurators', in *Proceedings of ECAI-2006 Workshop on Configuration*, volume 41, p. 44.

[7] Andreas Herrmann, Daniel G. Goldstein, Rupert Stadler, Jan R. Landwehr, Mark Heitmann, Reto Hofstetter, and Frank Huber, 'The Effect of Default Options on Choice—Evidence from Online Product Configurators', *J. Retailing and Consumer Services*, **18**(6), 483–491, (2011).

[8] Arnaud Hubaux, Dietmar Jannach, Conrad Drescher, Leonardo Murta, Tomi Männistö, Krzysztof Czarnecki, Patrick Heymans, Tien N. Nguyen, and Markus Zanker, 'Unifying Software and Product Configuration: A Research Roadmap', in *Proc. Configuration Workshop (ConfWS)*, pp. 31–35, (August 2012).

[9] Arnaud Hubaux, Yingfei Xiong, and Krzysztof Czarnecki, 'A User Survey of Configuration Challenges in Linux and eCos', in *Proc. Int'l Workshop Variability Modelling of Software-Intensive Systems (VaMoS)*, pp. 149–155, New York, NY, USA, (2012). ACM.

[10] Cynthia Huffman and Barbara E. Kahn, 'Variety for Sale: Mass Customization or Mass Confusion?', *J. Retailing*, **74**(4), 491–513, (1998).

[11] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, and Gunter Saake, 'An Overview on Analysis Tools for Software Product Lines', in *Proc. Workshop Software Product Line Analysis Tools (SPLat)*, pp. 94–101, New York, NY, USA, (2014). ACM.

[12] Frits K. Pil and Matthias Holweg, 'Linking Product Variety to Order-Fulfillment Strategies', *Interfaces*, **34**(5), 394–403, (October 2004).

[13] Giovani Da Silveira, Denis Borenstein, and Flávio S. Fogliatto, 'Mass Customization: Literature Review and Research Directions', *Int'l J. Production Economics*, **72**(1), 1–13, (2001).

[14] Clarissa Streichsbier, Paul Blazek, Fabian Faltin, and Wolfgang Frühwirt, 'Are De-Facto Standards a Useful Guide for Designing Human-Computer Interaction Processes? The Case of User Interface Design for Web Based B2C Product Configurators', in *Proc. Hawaii Int'l Conf. System Sciences (HICSS)*, pp. 1–7, Washington, DC, USA, (2009). IEEE.

[15] Juha Tiihonen and Alexander Felfernig, 'An Introduction to Personalization and Mass Customization', *J. Intell. Inf. Syst.*, **49**(1), 1–7, (August 2017).

# Configuring Release Plans

**A. Felfernig**[1] and **J. Spöcklberger**[1] and **R. Samer**[1] and **M. Stettinger**[1] and **M. Atas**[1] and **J. Tiihonen**[2] and **M. Raatikainen**[2]

**Abstract.** Release planning takes place (1) on the *strategic level* where the overall goal is to prioritize (high-level) requirements and (2) on the *operational level* where the major focus is to define more detailed implementation plans, i.e., the assignment of requirements to specific releases and often the assignment of stakeholders to requirements. In this paper, we show how release planning can be represented as a configuration task and how re-configuration tasks can be supported. Thus we advance the state-of-the-art in software release planning by introducing technologies that support the handling of inconsistencies in already existing plans.

## 1 Introduction

Higher flexibility of software development and better satisfied customer requirements can be achieved by developing and delivering software in an incremental fashion [8]. Release planning is needed to support such development approaches in a structured fashion. Release planning can be regarded as a company-wide optimization problem where stakeholders want to maximize the utilization of often limited resources [8, 10, 11]. Planning often takes place on two levels [1]. First, on a *strategic level* where the major task is to prioritize requirements with regard to criteria such as business relevance (profit), feasibility (risk)[3], and related efforts [9, 11]. On an *operational level*, a detailed planning takes place where requirements are assigned to releases and often also to stakeholders in charge of their implementation. The consequences of poor release planning are low software quality, lost business oportunities, more replanning efforts, and also project cancellation [10].

Figure 1 depicts an overview of a release planning process. First, requirements are prioritized on the basis of a utility analysis [3]. Second, detailed planning takes place where requirements are assigned to releases and a release planner is in charge of assuring the consistency of the plan with regard to a set of additional constraints related to dependencies between requirements and further constraints imposed by stakeholders (see the example release constraints depicted in Table 5).

The major contributions of this paper are the following. First, we show how to represent a release planning problem as a configuration task. In this context, we also show how reconfiguration can be supported on the basis of configuration and diagnosis techniques. Second, we report the results of a performance analysis that has been conducted on the basis of different types of release planning (configuration) problems.

The remainder of this paper is organized as follows. In Section 2, we sketch how utility analysis can be performed on the basis of a given set of requirements. Thereafter, in Section 3 we show how release planning can be represented as a configuration task and how re-configuration can be supported in this context. In Section 4, we report the results of an evaluation of the proposed approach. The paper is concluded with a discussion of issues for future work.

## 2 Utility-based Prioritization of Requirements

The prioritization of requirements can be performed on the basis of a utility analysis, i.e., the evaluation and ranking of requirements with regard to a predefined set of interest dimensions such as *profit*, *effort*, and *risk* [3]. In the line of the two basic recommendation approaches in group recommender systems [4], prioritization can be performed in two ways (see Figure 1): (1) *aggregated utilities* based approaches collect stakeholder-individual requirements evaluations with regard to a set of interest dimensions, aggregate those evaluations, and calculate requirement utilities thereof, (2) *aggregated prioritizations* based approaches aggregate stakeholder-specific prioritizations into the final prioritization. In the following, we explain both approaches in more detail. The second variant assumes that each stakeholder provides a prioritization (directly or in terms of utility evaluations) wheres the first approach also allows prioritization in situations where not all stakeholders evaluated each of the given requirements.

*Prioritization with Aggregated Utilities.* In this context, multi-attribute utility theory can be applied to determine a ranking (prioritization) of a given set of requirements (see Table 1). First, each stakeholder $s_i$ evaluates each individual requirement with regard to interest dimensions. In our example, we use the interest dimensions *profit*, *effort*, and *risk*, which are typical interest dimensions in release planning. Interest dimensions can have an assigned weight, for example, it is more important to avoid risky release plans than maximizing the potential profit.

In such a setting, the distribution of weights could be similar to the one defined in Table 2. Second, on the basis of a given set of evaluations and a specification of the importance of individual interest dimensions, requirements can be ranked according to Formula 1 where $imp(d)$ denotes the importance of interest dimension $d$ and $contrib(r, d)$ denotes the contri-

[1] Graz University of Technology, Graz, Austria email: {felfernig,spoecklberger,samer,stettinger,atas}@ist.tugraz.at
[2] University of Helsinki, Finland email: {juha.tiihonen,mikko.raatikainen}@helsinki.fi
[3] We interpret *profit* as the *value* of a requirement for the user [7].

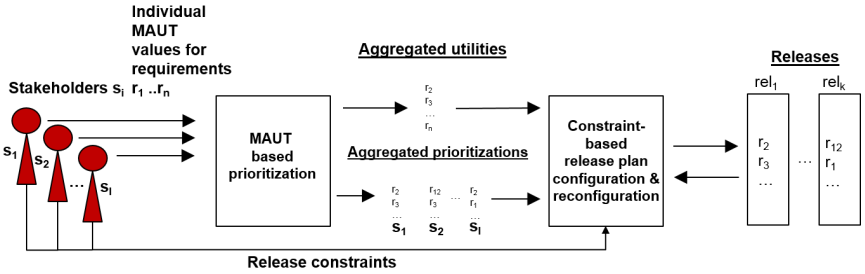


**Figure 1.** Utility-based prioritization and constraint-based configuration of release plans. Requirements can be prioritized on the basis of a group-based utility analysis (MAUT – multi-attribute utility theory [3]). The resulting prioritization can be determined on the basis of *aggregated utilities* or *aggregated prioritizations*. Release plans are generated on the basis of a given prioritization and corresponding release constraints defined by stakeholders.

bution of requirement $r$ to dimension $d$.

$$utility(r) = \Sigma_{d \in Dim} imp(d) \times contrib(r, d) \quad (1)$$

When applying Formula 1 to the entries in Table 1 and Table 2, we are able to derive a ranking of the requirements $R = \{r_1, r_2, .., r_n\}$ as depicted in Table 3.

In this paper, we directly evaluate requirements with regard to interest dimensions (on a rating scale [1..10]). Especially for the dimensions effort and profit, alternative evaluation scales can be defined which are then mapped to a utility scale (e.g., [1..10]). For example, instead of evaluating effort directly on a scale [1..10], effort could be specified in man-months which are then translated into a corresponding utility scale. In the context of our examples, high values for *profit* denote a high associated profit, whereas high values for *effort* and *risk* denote low associated effort and risk estimates.

*Prioritization with Aggregated Prioritizations.* First, multi-attribute utility theory can be applied to determine stakeholder-individual requirement utilities (priorities) (see Table 4). Each stakeholder $s_i$ evaluates individual requirements with regard to the pre-defined interest dimensions. Alternatively, stakeholders can specify prioritizizations "directly", i.e., without a utility-based pre-evaluation. Second, requirement utilities can be determined on the basis of Formula 2 where $s$ represents a stakeholder, $d \in Dim$ represents an interest dimension, and $r$ is a requirement. We assume a globally defined specific weight for the individual interest dimensions (see Table 2).

$$utility(r, s) = \Sigma_{d \in Dim} imp(d) \times contrib(r, d, s) \quad (2)$$

Stakeholder-individual prioritizations (see Table 4) have to be aggregated. One approach to support this aggregation is to apply basic social choice based preference aggregation functions such as *Borda Count* where the winner is the requirement with the best total ranking score where each requirement rank is associated with a score 0 .. #requirements-1 (see Table 6).[4]

*Testing Prioritizations.* A prioritization derived on the basis of a utility analysis can be tested for plausibility. For example, stakeholders can specify specific prioritization constraints that have to hold in the final prioritization. Such tests could be applied, for example, when different departments are cooperating in a prioritization process and specific constraints have been pre-defined by upper-level management. Such constraints can be regarded as test cases for the prioritization process (see *Definition 1*).

*Definition 1: Consistent Prioritization*: given a prioritization $P = \{p_1 : r_1 = v_1, p_2 : r_2 = v_2, .., p_n : r_n = v_n\}$ for requirements $REQ = \{r_1, r_2, .., r_n\}$ ($v_i \in domain(r_i)$) and a set of test cases $T = \{t_1, t_2, .., t_k\}$. Then $P$ is a consistent prioritization if $P \cup t_i$ is consistent $\forall t_i \in T$.

Consistent prioritizations determined on the basis of a utility analysis can be regarded as input for release planning. In the following, we show how prioritizations can be exploited in the context of release planning and how release planning can be represented as a constraint-based configuration task.

## 3 Constraint-based Release Configuration

Constraints $r_i = v_i$ ($v_i \in domain(r_i)$) representing individual requirement prioritizations can be used as one input of a release configuration problem [5] by generating release assignment constraints following the rule $\forall \{p_i : r_i = v_i, p_j : r_j = v_j\} \in P (i \neq j) : v_i > v_j \rightarrow rel_{rj} \geq rel_{ri}$. Since $r_1 > r_2$ holds in our working example, we can derive the constraint $pc : rel_{r2} \geq rel_{r1}$ as an input for our release configuration task (see Definition 2). We denote the set $\bigcup pc_i$ derived from a prioritization $P$ as $PC$.

*Definition 2: Constraint-based Release Configuration Task*: a constraint-based release configuration task can be defined as a tuple $(R, PC, RC)$ where $R = \{rel_{r1}, rel_{r2}, .., rel_{rn}\}$ is a set of variables representing potential *assignments of requirements to releases* ($domain(rel_{ri})$=[0..n] – 0 refers to requirements not assigned to a release), $PC = \{pc_1, pc_2, .., pc_m\}$ represents a set of *prioritization constraints*[5], and $RC = \{rc_1, rc_2, .., rc_l\}$ is a set of *release constraints*.

Examples of typically used release constraints are given in Table 5. Further release constraints that will not be taken into account in our working example are *release capacity in hours*, *total capacity in hours*, *release costs*, *total costs*, *assignment of stakeholders to requirements*, *average risk level per release*, *minimum market value per release*, and further optimization

[4] For an overview of different types of preference aggregation functions we refer to [4].

[5] Hard prioritization constraints are often too strict in practice. They can also be interpreted as preferences, i.e., solvers should take these into account as much as possible.

| requirement | $r_1$ | | | $r_2$ | | | $r_3$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | profit | effort | risk | profit | effort | risk | profit | effort | risk |
| $s_1$ | 3 | 3 | 4 | 7 | 8 | 6 | 1 | 2 | 1 |
| $s_2$ | 5 | 2 | 4 | 4 | 4 | 5 | 3 | 4 | 3 |
| $s_3$ | 6 | 3 | 2 | 5 | 5 | 7 | 4 | 1 | 4 |
| $average(AVG)$ | 4.67 | 2.67 | 3.33 | 5.33 | 5.67 | 6.0 | 2.67 | 2.33 | 2.67 |

**Table 1.** Contribution (on a scale 1–10) of requirements $R = \{r_1, r_2, r_3\}$ to the interest dimensions $Dim = \{profit, effort, risk\}$. Following the *"aggregated utilities"* approach, utility analysis determines a prioritization. $AVG$ is one possible aggregation function – for further alternatives we refer to [4].

| | profit | effort | risk |
|---|---|---|---|
| $importance(imp)$ | 1 | 3 | 6 |

**Table 2.** Importance of interest dimensions in a specific requirements prioritization context.

| | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| $utility(r_i)$ | 32.66 | 58.34 | 25.68 |
| $prioritization$ | 2 | 1 | 3 |

**Table 3.** Utility and prioritization of individual requirements $R = \{r_1, r_2, r_3\}$ with regard to the interest dimensions $Dim = \{profit, effort, risk\}$. In this context, we assume that each requirement has its unique prioritization, i.e., prioritization$(r_i)$ = prioritization$(r_j) \rightarrow i = j$.

criteria such as *maximum profit*, *maximum customer value*, and *minimum risk*. For simplicity, Table 5 contains only binary constraints, however, these are generalizable to higher-order constraints [2], for example, coupling$(rel_{ra}, rel_{rb}, rel_{rc})$ would be translated into $rel_{ra} = rel_{rb} \wedge rel_{rb} = rel_{rc}$.

An example of a constraint-based release configuration task is the following. This task includes the three requirements from Section 2. Furthermore, three releases are available.

- $R = \{rel_{r1}, rel_{r2}, rel_{r3}\}$
- $domain(rel_{ri}) = [1..3]$
- $PC = \{pc_1 : rel_{r2} \leq rel_{r1}, pc_2 : rel_{r2} \leq rel_{r3}, pc_3 : rel_{r1} \leq rel_{r3}\}$
- $RC = \{rc_1 : rel_{r_1} = rel_{r_2}, rc_2 : rel_{r1} = 1\}$

*Definition 3: Constraint-based Release Configuration*: A constraint-based release configuration (solution) for a constraint-based release configuration task can be represented by a complete assignment $RP = \{rel_{ra} = va, rel_{rb} = vb, .., rel_{rk} = vk\}$ where $vk$ is the release number of requirement $rk$ such that $RP \cup PC \cup RC$ is consistent.

In the context of our working example, an example of a constraint-based release configuration is $RP = \{rel_{r1} = 1, rel_{r2} = 1, rel_{r3} = 2\}$.

As it is often the case, prioritization as well as release planning is an iterative process [8]. As a consequence, prioritizations $PC$ of requirements change (resulting in PC') as well as release constraints, i.e., $RC$ is transformed into $RC'$. In such contexts, situations can occur where $RP \cup PC' \cup RC'$ becomes inconsistent and we are in the need of a reconfiguration of $RP$ (resulting in $RP'$). Consequently, a release reconfiguration task has to be solved (see Definition 4).

*Definition 4: Release Reconfiguration Task*: A release reconfiguration task can be defined by a tuple $(RP, PC', RC')$ where $PC'$ represents a set of adapted prioritization constraints, $RC'$ a set of adapted release constraints, and $RP \cup PC' \cup RC'$ is assumed to be inconsistent. The underlying task is to identify a minimal set of constraints $\Delta \subseteq RP$ such that $RP - \Delta \cup PC' \cup RC'$ is consistent. If parts of $RP$ have already been implemented, we can assume $RP = RP_{completed} \cup RP_{open}$ and the task is to identify a diagnosis $\Delta$ with $RP_{open} - \Delta \cup RP_{completed} \cup PC' \cup RC'$ is consistent.

A reconfiguration for a given release reconfiguration task can be defined as follows (see Definition 5).

*Definition 5: Release Reconfiguration*: A release reconfiguration (solution) for a release reconfiguration task can be represented by an assignment $RECONF = \{rel_{ra} = va', rel_{rb} = vb', .., rel_{rk} = vk'\}$ where $rel_{ri} = vi' \in RECONF \rightarrow rel_{ri} = vi \in \Delta$ and $vi \neq vi'$.

In this context, $\Delta$ represents a diagnosis, i.e., a minimal set of constraints that, if deleted from $RP$ ($RP_{open}$), help to restore consistency, i.e., $RP - \Delta \cup PC' \cup RC'$ is consistent.

The set $\Delta$ can be determined on the basis of a model-based diagnosis algorithm such as FastDiag [6] which returns a minimal set of constraints that have to be deleted in order to restore consistency. In order to assure the existence of a diagnosis $\Delta$, we have to assume the consistency of $PC' \cup RC'$.

One could also be interested in identifying minimal sets of changes $\Delta$ in given prioritizations $PC$ such that $PC - \Delta \cup RC$ is consistent. Table 7 provides an overview of example (re-)configuration services that can be provided in release configuration scenarios. (1) proposed prioritizations ($PC$) have to be checked with regard to their consistency with a defined set of release constraints ($RC$). (2) Assuming the consistency of $RC$

| requirement | $r_1$ | | | | $r_2$ | | | | $r_3$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | profit | effort | risk | utility (prio) | profit | effort | risk | utility (prio) | profit | effort | risk | utility (prio) |
| $s_1$ | 3 | 3 | 4 | 36 (2) | 7 | 8 | 6 | 68 (1) | 1 | 2 | 1 | 13 (3) |
| $s_2$ | 5 | 2 | 4 | 35 (2) | 4 | 4 | 5 | 46 (1) | 3 | 4 | 3 | 33 (3) |
| $s_3$ | 6 | 3 | 2 | 27 (3) | 5 | 5 | 7 | 62 (1) | 4 | 1 | 4 | 31 (2) |

**Table 4.** Contribution (on a scale 1–10) of requirements $R = \{r_1, r_2, r_3\}$ to the interest dimensions $Dim = \{profit, effort, risk\}$. High values for profit denote a high associated profit, whereas high values for effort and risk denote low associated effort and risk estimates. Following the *"aggregated predictions"* approach, utility analysis determines stakeholder-specific prioritizations.

| constraint | formalization | description |
|---|---|---|
| coupling($rel_{ra}, rel_{rb}$) | $rel_{ra} = rel_{rb}$ | $\{rel_{ra}, rel_{rb}\}$ have to be implemented in the same release |
| different($rel_{ra}, rel_{rb}$) | $rel_{ra} \neq rel_{rb} \vee rel_{ra} = 0 \wedge rel_{rb} = 0$ | $\{rel_{ra}, rel_{rb}\}$ have to be implemented in different releases |
| eitheror($rel_{ra}, rel_{rb}$) | $(rel_{ra} = 0 \wedge rel_{rb} \neq 0) \vee (rel_{ra} \neq 0 \wedge rel_{rb} = 0)$ | $\{rel_{ra}, rel_{rb}\}$ are exclusive |
| atleastone($rel_{ra}, rel_{rb}$) | $\neg(rel_{ra} = 0 \wedge rel_{rb} = 0)$ | at least one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to a release |
| atleastonea($rel_{ra}, rel_{rb}, a$) | $\neg(rel_{ra} \neq a \wedge rel_{rb} \neq a)$ | at least one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to release $a$ |
| atmostone($rel_{ra}, rel_{rb}$) | $\neg(rel_{ra} \neq 0 \wedge rel_{rb} \neq 0)$ | at most one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to a release |
| atmostonea($rel_{ra}, rel_{rb}, a$) | $\neg(rel_{ra} = a \wedge rel_{rb} = a)$ | at most one out of $\{rel_{ra}, rel_{rb}\}$ has to be assigned to release $a$ |
| weakprecedence($rel_{ra}, rel_{rb}$) | $rel_{ra} \leq rel_{rb}$ | $rel_{ra}$ must be implemented before $rel_{rb}$ or in the same release |
| weakprecedence($rel_{ra}, rel_{rb}$) | $rel_{ra} \leq rel_{rb} \wedge rel_{ra} > 0$ | $rel_{ra}$ must be implemented before $rel_{rb}$ or in the same release |
| strictprecedence($rel_{ra}, rel_{rb}$) | $rel_{ra} < rel_{rb}$ | $rel_{ra}$ must be implemented before $rel_{rb}$ |
| strictprecedence($rel_{ra}, rel_{rb}$) | $rel_{ra} < rel_{rb} \wedge rel_{ra} > 0$ | $rel_{ra}$ must be implemented before $rel_{rb}$ |
| valuedependency($rel_{ra}, rel_{rb}$) | $\neg(\lvert rel_{ra} - rel_{rb}\rvert > k)$ | $rel_{ra}$ and $rel_{rb}$ must be implemented in nearly the same release |
| effortdependency($rel_{ra}, rel_{rb}$) | $\neg(\lvert rel_{ra} - rel_{rb}\rvert > k)$ | $rel_{ra}$ and $rel_{rb}$ must be implemented in nearly the same release |
| fixed($rel_r, a$) | $rel_r = a$ | requirement $r$ must be implemented in release $a$ |
| nolaterthan($rel_r, a$) | $rel_r \leq a$ | requirement $r$ must not be implemented after release $a$ |
| notearlierthan($rel_r, a$) | $rel_r \geq a$ | requirement $r$ must not be implemented before release $a$ |

**Table 5.** Example release constraints. For the constraint types *valuedependency* and *effortdependency* we assume a maximum deviation of $k$.

| | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| $s_1$ | 2 | 1 | 3 |
| $s_2$ | 3 | 1 | 2 |
| $s_3$ | 3 | 1 | 2 |
| *score* ($BRC$) | 1 | 6 | 2 |
| *prioritization* | 3 | 1 | 2 |

**Table 6.** Aggregation of stakeholder-individual prioritizations based on the *Borda Count* ($BRC$) aggregation function (highest score receives 2 points, second highest score 1 point, and lowest score receives 0 points [4]).

and the inconsistency of $PC \cup RC$, a minimal set of elements in $PC$ has to be identified such that $PC - \Delta \cup RC$ is consistent. (3) Assuming the consistency of $PC' \cup RC'$, a minimal set of elements in $RP$ has to be identified (i.e., a potential change of the current release plan) such that $RP - \Delta \cup PC' \cup RC'$ is consistent. (4) Given a diagnosis $\Delta$ for $RP$ (with regard to $PC' \cup RC'$), a constraint solver can determine a solution for $RP - \Delta \cup PC' \cup RC'$. (5) If there exists a test case $t \in T$ with $inconsistent(t \cup PC)$, a diagnosis $\Delta$ represents a minimal set of elements in $PC$ such that $PC - \Delta \cup t$ is consistent $\forall t \in T$.

| ID | service |
|---|---|
| 1 | consistency check of $PC \cup RC$ |
| 2 | diagnosis of $PC$ with regard to $RC$ |
| 3 | diagnosis of $PC$ with regard to test cases $T$ |
| 4 | diagnosis of $RP$ with regard to $PC' \cup RC'$ |
| 5 | reconfiguration of $RP$ with regard to $PC' \cup RC'$ |
| 6 | diagnosis of $RC$ |

**Table 7.** Overview of example release (re-)configuration services.

## 4 Evaluation

Beside performance analyses, there are different alternative ways to evaluate the release planning related services mentioned in Table 7.

*Release plans* as outcome of a configuration process can be evaluated with regard to different interest dimensions such as profit, risk, and effort. The corresponding utility function is the following (see Formula 3).

$$utility(RP) = \frac{\Sigma_{rel_r \in RP} \Sigma_{d \in Dim} \frac{contrib(r,d) \times imp(d)}{rel_r}}{|rel_r \in RP|} \quad (3)$$

An evaluation of the utility of release plans generated with the Choco constraint solver[6] is depicted in Table 8. A corresponding performance evaluation is depicted in Table 9. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints 10 times. In this context, we did not optimize solution search on the basis of search heuristics – this is regarded as a major task of our future work. In Table 8, we can observe a decreasing utility of release plans along with an increasing size of $RC$. Table 9 shows increasing runtimes with an increasing size of $RC$ and an increasing number of releases.

[6] choco-solver.org

| $\|RC\|$ | #releases | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 5 | 10 | 25 | 50 | 100 |
| 25 | 114.6 | 110.8 | 123.3 | 123.2 | 114.2 | 118.1 |
| 50 | 125.3 | 110.9 | 118.0 | 121.9 | 112.9 | 120.7 |
| 100 | 114.4 | 99.6 | 111.3 | 108.0 | 111.3 | 120.9 |
| 250 | 114.3 | 78.4 | 827.3 | 104.2 | 112.0 | 114.9 |
| 500 | 123.1 | 61.8 | 70.6 | 88.4 | 110.6 | 123.1 |
| 1000 | 112.3 | 63.1 | 50.7 | 71.7 | 95.3 | 109.4 |

**Table 8.** Avg. utility of release plans without optimization.

| $\|RC\|$ | #releases | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 5 | 10 | 25 | 50 | 100 |
| 25 | 54.0 | 19.9 | 16.5 | 77.2 | 142.5 | 587.9 |
| 50 | 36.1 | 22.1 | 33.3 | 53.5 | 142.5 | 622.4 |
| 100 | 75.0 | 84.1 | 85.0 | 144.1 | 654.2 | 641.9 |
| 250 | 300.8 | 721.9 | 829.4 | 1161.5 | 2094.2 | 3831.6 |
| 500 | 1110.3 | 2824.9 | 5053.4 | 6790.1 | 9372.7 | 16677.6 |
| 1000 | 5018.1 | 12537.6 | 23975.8 | 43738.5 | 58207.6 | 73785.1 |

**Table 9.** Avg. performance of release plan determination in *ms*.

*Reconfigurations* of release plans can be evaluated with regard to the *similarity* between the new configuration and the old configuration where $a(S)$ denotes the set of variable assignments contained in solution $S$.

$$similarity(S_{old}, S_{new}) = \frac{a(S_{old}) \cap a(s_{new})}{a(S_{old}) \cup a(s_{new})} \quad (4)$$

An evaluation of the similarity between reconfigurations and original release plans is depicted in Table 10. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints and a corresponding release plan 10 times, randomly changed 10% of the (original) release constraints and determined a reconfiguration (for the given release plan). We can observe a decreasing similarity with a corresponding increasing number of release constraints.

| $\|RC\|$ | #releases | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 5 | 10 | 25 | 50 | 100 |
| 25 | .96 | .99 | 1.00 | 1.00 | 1.00 | 1.00 |
| 50 | .93 | .97 | .99 | 1.00 | 1.00 | 1.00 |
| 100 | .87 | .96 | .97 | .99 | .99 | 1.00 |
| 250 | .96 | .87 | .93 | .98 | .99 | .99 |
| 500 | .99 | .76 | .86 | .95 | .97 | .99 |
| 1000 | 1.00 | .75 | .78 | .90 | .95 | .97 |

**Table 10.** Avg. similarity of reconfigurations.

*Diagnoses* can be evaluated with regard to their degree of *conservativism* (see Formula 5), i.e., the number of changes needed in a constraint set $C$ compared to the overall number of elements in $C$. Furthermore, *diagnoses* can be evaluated with regard to their *relevance*: the lower the total relevance of constraints contained in a diagnosis (represented as the sum of the individual relevances ($rel(\delta_i)$) of constraints in $\Delta$), the higher the relevance of the $\Delta$ (see Formula 6).

$$conservativism(\Delta, C) = 1 - \frac{|\Delta|}{|C|} \quad (5)$$

$$relevance(\Delta = \{\delta_1, \delta_2, .., \delta_q\}) = \frac{\Sigma_{\delta_i \in \Delta} rel(\delta_i)}{|\Delta|} \quad (6)$$

An evaluation of conservativism and relevance of generated diagnoses is depicted in Table 11. For each combination of $|RC| \times \#releases$, we randomly generated $|RC|$ release constraints 10 times, assigned importance values to these constraints, and randomly changed 10% of the constraints. The resulting (inconsistent) constraint sets were diagnosed with FastDiag [6]. The corresponding evaluation results are depicted in Table 11. We can observe a decreasing degree of conservativism with an increasing number of release constraints $RC$.

| | #releases | | | | | |
|---|---|---|---|---|---|---|
| $\|RC\|$ | 1 | 5 | 10 | 25 | 50 | 100 |
| 25 | .84/1.0 | .99/.2 | .99/.1 | 1.00/0 | 1.00/0 | 1.00/0 |
| 50 | .75/1.0 | .99/.5 | .99/.4 | 1.00/0 | 1.00/0 | 1.00/0 |
| 100 | .62/1.0 | .98/1.0 | .99/.7 | 1.00/.2 | 1.00/0 | 1.00/0 |
| 250 | .58/1.0 | .86/1.0 | .96/1.0 | .99/.9 | 1.00/.8 | 1.00/.1 |
| 500 | .56/1.1 | .76/1.0 | .89/1.0 | .98/1.0 | .99/.8 | 1.00/.8 |
| 1000 | .55/1.2 | .64/1.0 | .76/1.0 | .92/1.0 | .97/1.0 | .99/1.0 |

**Table 11.** Avg. conservativism and relevance (c/r) of diagnoses.

## 5 Conclusion and Future Work

In this paper, we have shown how to represent release planning as a configuration problem. This representation is a major basis for supporting re-planning tasks, i.e., the adaptation of plans that become inconsistent due to changing constraints (e.g., a changing availability of resources). In this context, we also focused on introducing concepts that support the automated adaptation of release plans (reconfiguration of release plans) and different variants thereof such as the diagnosis of release constraints and prioritization constraints. To show the applicability of the presented concepts, we have presented initial evaluation results based on a couple of evaluation metrics. Future work will include the development and evaluation of different types of preference aggregation functions (see Section 2) with regard to their capability of generating relevant prioritizations. Furthermore, we will optimize the determination of release plans, reconfigurations, and diagnoses by integrating intelligent search heuristics that help to improve the quality of identified solutions. In this context, we will compare constraint-based reasoning approaches with local search based ones [8] with regard to efficiency and solution quality.

## Acknowledgment

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OpenReq (732463).

## REFERENCES

[1] D. Ameller, C. Farre, X. Franch, D. Valerio, and A. Cassarino, 'Towards continuous software release planning', in *SANER 2017*, pp. 402–406, Klagenfurt, Austria, (2017).

[2] F. Bacchus, X. Chen, P. van Beek, and T. Walsh, 'Binary vs. non-binary constraints', *Artificial Intelligence*, **140**(1–2), 1–37, (2002).

[3] J. Dyer, 'Multi attribute utility theory', *International Series in Operations Research and Management Science*, **78**, 265–292, (1997).

[4] A. Felfernig, L. Boratto, M. Stettinger, and M. Tkalcic, *Group Recommender Systems – An Introduction*, Springer, 2018.

[5] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 1st edn., 2014.

[6] A. Felfernig, M. Schubert, and C. Zehentner, 'An Efficient Diagnosis Algorithm for Inconsistent Constraint Sets', *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, **26**(1), 175–184, (2012).

[7] D. Greer, D. Bustard, and T. Sunazuka, 'Prioritization of system changes using cost-benefit and risk assessments', in *4th International Symposium on Requirements Engineering*, pp. 180–187, Limerick, Ireland, (1999).

[8] D. Greer and G. Ruhe, 'Software release planning: An evolutionary and iterative approach', *Information and Software Technology*, **46**(4), 243–253, (2004).

[9] H. Jung, 'Optimizing value and cost in requirements analysis', *IEEE Software*, **15**(4), 74–78, (1998).

[10] M. Lindgren, R. Land, C. Norström, and A. Wall, 'Key aspects of software release planning in industry', in *19th Australian Conference on Software Engineering*, pp. 320–329, Perth, WA, Australia, (2008).

[11] G. Ruhe and M. Saliu, 'The art and science of software release planning', *IEEE Software*, **22**(6), 47–53, (2005).

# Insights for Configuration in Natural Language

**Andrés F. Barco**[1] and **Élise Vareilles**[2] and **César I. Osorio**[1]

**Abstract.** Usually, in configuration processes, customers interact with a decision support system, also named configurator, by explicitly selecting components or required functionalities through a written series of questions, until the complete configuration is done and the desired product is defined. The interactions during a configuration process may vary vastly depending on the customers' knowledge about the product and his/her understanding of its potential functionalities. However, configurators are not conceived for making a difference between expert and inexpert customers as interfaces and input information are all expected to be the same for everyone. This paper discusses how *natural language* can enhance configuration process by making possible for customers to express their desires, needs and preferences in natural language, and for configurators to interpret their words and better help them to find relevant solutions. This kind of configuration process could have as foundations an expert systems that that maps speech into constraints and objectives. We present the artificial intelligence trends motivating our research, an initial architectural design and potential applications of the research.

## 1 Introduction

For several decades now, customers want to bring a personal touch to their products to make them special and unique. To meet this growing demand of personalization, companies nowadays no longer offer standard products, but more and more personalizable ones [1]. Thanks to the Web technologies and dedicated decision support systems, named configurators, this personalization is done directly and interactively online [1]. Customers can play with the wide range of choices and options offered by companies: they can assemble, cut, color, choose, ..., and visualize the result of their desires and ultimately order it, all in few clicks and minutes.

This concept of personalization or configuration of products consists in assembling modules or predefined components, to produce a unique and specific product [10]. For businesses, this is a way to offer personalized products to stand out from the competition and build customers' loyalty through more accurately reflecting their tastes and needs.

Interactions between potential customers and configurators, become now one of the key aspects of configuration problems [22]. Nevertheless, often the configuration relies in a long data capturing process. Normally, to configure the product object of desires, any potential customer has to:

1. Face the increasing range of choices and options without being completely able to focus on his/her essential items,
2. answer a predefined series of questions, always in the same order whatever his/her knowledge and needs about the product,
3. express his/her needs and preferences in such a way they fit the predefined set of choices and options proposed online by configurators, and
4. click to select the relevant functions or components meeting the best of his/her needs.

All these facts gathered make the configuration of products more and more a counter-intuitive process. Also, current interaction makes no difference between expert and non-expert user as most of the input mechanism (such of as graphical windows, drawings, text fields) are all expected to be the same for everyone [2].

This paper discusses how the mature techniques from AI may be used to allow a more natural interaction between customers and configuration systems. In essence, we describe the future of configuration in which natural language interactions could replace the traditional one based on writing, explicit selection of items and rigid series of questions. More precisely, we argue that current AI advances like natural language processing could help customers to express their needs implicitly in *speech* format (e.g. "I need a cheap laptop with good video card") while an expert system could infer the requirements and set the goals of the configuration (e.g. video card $\geq$ good). We call this kind of configuration *Configuration in Natural Language*. We also present an idea of architecture of such an expert system.

The paper is divided as follows. In Section 2, the traditional and future customer-system interactions in configuration are discussed. In section 3, a software architecture based on current advances in AI is introduced. In Section 4, some of the mathematical frameworks from which the configuration in natural language can take advantage are discussed. Finally, in Section 5, some applications of the research and conclusions are presented.

## 2 Configuration Interactions: Past and Future

We present in this section the interaction typically made in configuration systems and contrast it with the idea of configuration in natural language proposed in this paper.

### 2.1 Traditional Configuration Interactions

In most of the cases, configuration systems follow a series of iterative steps that guide the customer and help him/her to progressively configure his/her own product. These steps, that we call the *standard way of configuration*, are as follows:

1. The system shows some sets of components and functionalities to the customer in a predefined way,

[1] Universidad de San Buenaventura Cali. Santiago de Cali, Colombia. email: {anfelbar,ciosoriod}@usbcali.edu.co
[2] Université de Toulouse, Mines Albi. Albi, France. email: elise.vareilles@mines-albi.fr

2. The customer either selects the most relevant component or functionality which meets his/her needs and desires, or specifies a value for the most important criteria (such as the price he/she is willing to pay for the product),
3. The system removes or assigns the set of remaining components and functionalities according to the set of constraints in the product,
4. The system computes the price of each possible product and valuates their criteria.

In step 2, the selection is usually done manually by clicking on the relevant item via a mouse-click or a touch screen, or by inputting its specific value directly from a keyboard. At the end of the standard way of configuration, the customer selects his/her unique product and orders it.

## 2.2 Future Configuration Interactions

The configuration in natural language changes the interactions between customers and configurators. Customers will be able, whatever their requirements and knowledge about the product, to express better in a more natural way their preferences, desires and needs and configure faster their own products. The significant difference between traditional and forthcoming configuration interactions lies in the way of expressing and capturing customers' needs and goals. The steps for the configuration in natural language are as follows:

1. The system welcomes the customer.
2. The customer writes or says what he/she wants or needs by using his/her own words.
3. The system infers the set of components and functionality the user wants or needs, and the goal of the configuration.
4. The system removes or assigns the set of remaining components and functionalities according to the set of constraints in the product.
5. The system computes the price of each possible product and valuates their criteria.

To exemplify this kind of configuration, limit us to written interactions in natural language via a keyboard or similar device. Three examples of such interactions when configuring a computer, and the respective responses of the inferring engine, are:

- **Customer express:** "I want a really fast computer but not too expensive"
  **System infers:** *Component*(processor, speed, high) *Goal*(computer, cost, low)
- **Customer express:** "I just want to play video games, preferably not heavy so I can carry it easily"
  **System infers:** *Component*(video_card, processing, high) *Goal*(computer, weight, low)
- **Customer express:** "I need to write my texts"
  **System infers:** *Component*(keyboard, comfort, high) *Goal*(none, none, none)

As expected, inferring components or functionalities and configuration objectives is a major challenge. On the first hand, the universe of words used by humans to express the same thing may be vast. Second, the way to build expressions may vary largely depending on academic background, experience, state of mind and mood, to name a few. Finally, it is difficult to set a difference between components and functionalities, and goals; critical to reach an appropriate configuration solution. For instance, in the first of the two previous examples, both the processor speed and the computer cost may be seen as configuration goals. For tackling this challenge, we propose an expert system architecture built upon AI trends.

# 3 AI Trends-based Architecture

The field of AI is generating broad interest. Technological advances using AI techniques, such as the DeepMind GO system developed by Google™ [19] and the IBM Watson™ analytic system [11], draw the attention of the academic and non-academic world on the innovative role of mathematical models from computer science and philosophy. Current trends show that the use of AI and other related fields are being widely used sectors such as economy, health, transport industry, aviation and games, among others [20].

The configuration in natural language is motivated by the recent advances in AI and by industrial trends, in particular the growing interest in the construction of machines that understand human emotions and act according to the interaction with human [16]. It is sought that the machines assist decision-making processes whereas the understanding of human emotions helps to improve the expert systems behavior and interaction [8]. This idea, known as *Human Aware AI*, is not new in configuration as it has been used as a goal in different configuration systems (see for instance [3]). Further, the idea of understating or inferring user needs has been widely study in the plan recognition problem [6]. Nonetheless, to the best of our knowledge, current advances in natural language use remain to be adopted in configurators implementations.

Within the human aware AI field, the Natural Language Understating (NLU) [4] and Natural Language Processing (NLP) [12] draw attention for its capabilities of human computer interaction. In essence, NLU and NLP systems allow the user to ask questions in everyday language and try to *understand* these questions in order to return appropriated answers. Typically, these systems makes some hypothesis according to the question and a knowledge base, such as Internet, and then process an output. This is akin to the problem of plan recognition, i.e., knowing the user's plans and goals [6]. Further, if these systems are improved with natural language generation (NLG) in order to produce responses, the system then becomes a question answering system (QAS) [13]. These systems were conceived to received provide argued answers to user queries. From these systems, WolframAlpha™ [7] and IBM Watson™ [11] present the more innovative results for configuration as these systems are able to recognize some information in form of requirements within informal speech in text format.

To illustrate these capabilities, Figure 1 shows the result when querying *"I want a computer of less than 1000 dollars."* in the WolframAlpha™ system. To construct a response, the system maps the input into a more elaborated query, encoding the query thus making a syntactic and semantic analysis. Nevertheless, as the system does not focus on inferring mathematical notions, it is easily confused by adding words that add relevant constraints. For instance, no result shown when querying the same computer but Dell™ manufactured; *"I want a dell computer of less than 1000 dollars"*. We consider that this is a major drawback in the system when addressing configuration problems.

The IBM Watson™ system works similarly to the WolframAlpha™. It encodes a query by applying automated reasoning, machine learning and several other techniques to analyze the speech. One of the more innovative applications of the IBM

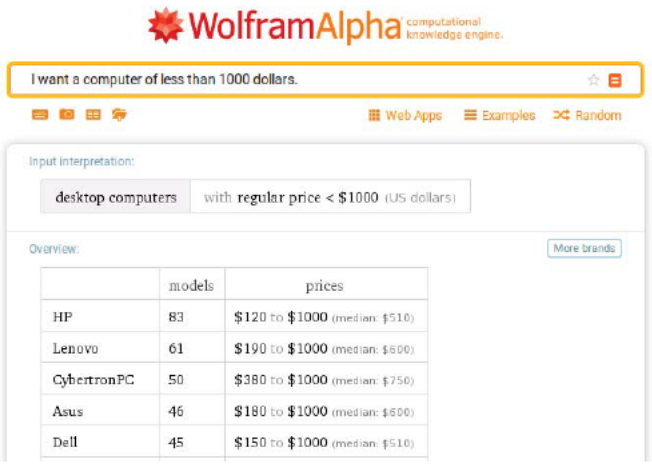


**Figure 1.** Output example of `WolframAlpha`™ system.

`Watson`™ system is the personality characterization from written speech. To do this, in addition to question answering, different techniques of sentiment analysis [21] are applied. This kind of analysis may be useful to infer expertise level of the user and then behave accordingly.

In spite of the recent advances of NLU, NLP and QAS, these are not well-suited for addressing configuration problems given that they do not focus on constructing the mathematical notions (constraints and objectives) needed in most configuration problems. Besides, question answering systems are too powerful in the sense they are of general purpose having different question types and extensive knowledge bases. At the other end of the spectrum, specific applications using question answering systems technology do not necessarily deal with extensive vocabularies, hypothesis and so on, as these applications are domain-dependent. Ergo, its underlying mechanism may be simpler although more robust and may count with reduced knowledge, question types and small knowledge bases. In consequence, we have devise an architecture, presented in Figure 2, for implementing configurators that exploits the aforementioned natural language elements. The mandatory module is that of NLU whereas NLP and NLG are optional (used if formatted answers are desired). External services may be attached in order to fulfill specific tasks.

The differentiating element in the architecture is the inference engine. This key element is in charge of discriminating among the set of words those referring to components of the product, functionalities and/or configuration objectives. This is in fact a challenge as different conclusions may be reached from a given sentence.

## 4 Mathematical Frameworks for Configuration in Natural Language

Unlike NLP and QAS, configuration processes are mapped, generally, into mathematical (optimization) models that unify the customers requirements and problem domain limitations into a single framework. To infer constraints and objectives from informal speech, it is needed an underlying mathematical framework in which such notions are built. In other words, to construct a mathematical formula it is necessary the set of values and operands allowed in the formula. As an example, if constructing inequalities, the expert system knows

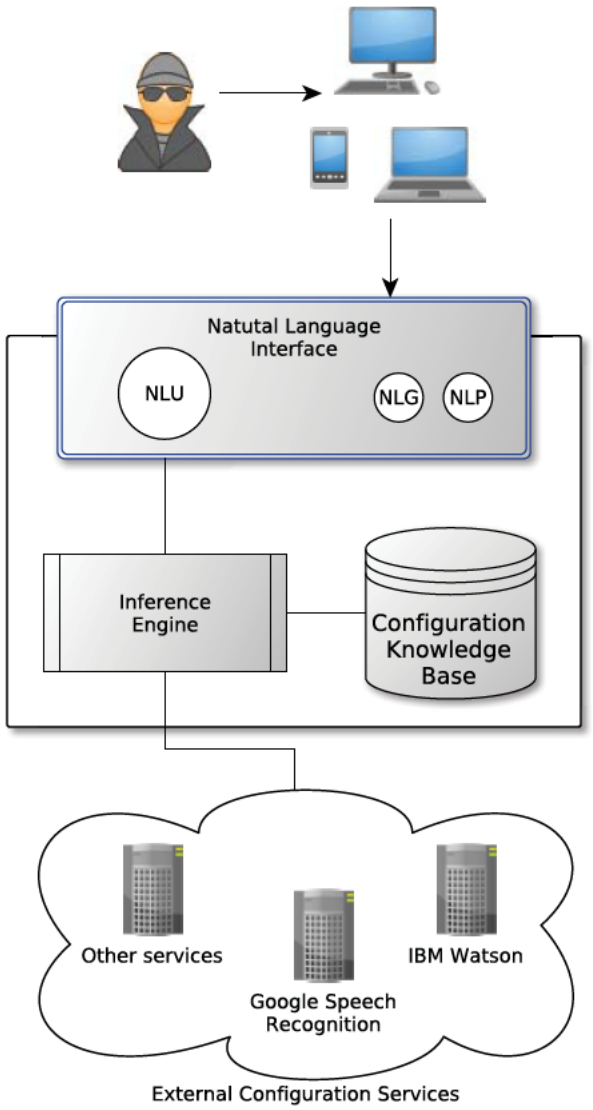


**Figure 2.** Architectural view of expert system for configuration NL.

that sum, subtract, division and multiplication are allowed as well as inequality symbols. This helps to make the mapping unambiguous. Given that our main innovative application is the configuration in natural language, mathematical frameworks used to tackle configuration problems comes naturally. Here, we briefly describe three of these models.

- The first framework, Constraint Programming (CP), has been identified as a key paradigm in the expansion of applied computer science [17]. CP is part and good representative, of the declarative programming frameworks. This is one of the most used framework to address configuration problems as it suits their constrained nature [15]. First, the knowledge (constraints) that restricts possible configuration of elements (variables) is easily modeled under the declarative framework of constraint satisfaction problems. And second, constraint-based configurators are able to present different

solutions to users, often optimal, even when they do not provide all configuration parameters.

- Among the same lines we have integer programming (IP). It is a mathematical optimization in which some or all of the variables are restricted to be integers [18]. Likewise the constraint satisfaction model, it is built from constraints and limitations over variables, although variables do not have a given domain, and objective functions such as minimization and maximization. This model or extensions of it (Mixed IP or Linear IP) have been widely used to reason and solve configuration problems (see for instance [9]).
- Finally, another framework that can be used in configuration in natural language is SAT; determining if a given propositional formula is satisfiable by an interpretation [5]. The inclusion or exclusion of a given feature in a product may be seen as boolean assignments. Thus, it would be possible to deduce if a solution exists by constructing a boolean formula using the requirements of the customers and the configuration knowledge (e.g. compatibility among components). The modeling of a configuration problem under SAT is not new (see for instance [14]).

## 5 Concluding Remarks

In this paper, we have presented insights on how to enhance the interaction between customers and configuration systems into something called configuration in natural language. In summary, we have discussed the possibility of using AI techniques, namely, human-aware AI to implement an expert system that infers constraints and objectives from a speech input.

We have built out the idea around the concepts of NLU, NLP and QAS. Then, we argue that although current systems have reached a stable development, there is still research to be done when inferring mathematical notions from informal sentences, i.e., everyday language. In addition, we have argued that advanced question answering systems such as `IBM Watson`™ are not yet well suited for configuration problems as they are not intended to map implicit requirements into mathematical models.

To give a view of the mathematical models that can prove useful to construct the proposed expert system, we have briefly described three techniques from AI and operational research. These, techniques, namely, constraints programming, integer programming and boolean satisfiability, have been used to solve different configuration and optimization problems and are in our point of view interesting models to construct a configurator in natural language.

The main application of inferring constraints and goals from speech is linked to the mathematical modeling of real-world problems. Simply stated, the inference system may be used to build mathematical models to solve linear problems, discrete optimization problems and probabilistic problems, among others. Further, specific properties from each mathematical framework may be exploited, such as the expressivity of declarative approaches like logic and constraint programming. Intuitively, many more applications exists; those in which user requirements, preferences, limitations or objectives are needed. For instance package managers in unix-based operating systems. Typically, a package manager from a Unix-based system must be asked to search or install a specific package. Using a expert systems that maps speech into constraints and objectives, a manager would accept inputs like *"I need a powerful UML diagram editor"* to present some potential editors to be installed. Further, arguments used by such managers can be replaced by everyday words thus preventing the non-expert user to learn the specifics of the programs.

## REFERENCES


[1] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, M. Meyer, G. Petrone, R. Schafer, W. Schutz, and M. Zanker, *Personalising on-line configuration of products and services*, 225–229, Thiel, S., Pohl, K., Lyon France, 2002.

[2] Liliana Ardissono, Anna Goy, Matt Holland, Giovanna Petrone, and Ralph Schäfer, *Customising the Interaction with Configuration Systems*, 283–287, Springer Berlin Heidelberg, 2003.

[3] Virginia E. Barker, Dennis E. O'Connor, Judith Bachant, and Elliot Soloway, 'Expert systems for configuration at digital: Xcon and beyond', *Commun. ACM*, **32**(3), 298–318, (March 1989).

[4] M Bates, 'Models of natural language understanding', *Proceedings of the National Academy of Sciences*, **92**(22), 9977–9982, (1995).

[5] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2009.

[6] Sandra Carberry, 'Techniques for plan recognition', *User Modeling and User-Adapted Interaction*, **11**(1), 31–48, (Mar 2001).

[7] John B. Cassel, *Wolfram—Alpha: A Computational Knowledge "Search" Engine*, 267–299, Springer New York, New York, NY, 2016.

[8] Celso M. de Melo, Stacy Marsella, and Jonathan Gratch, '"do as i say, not as i do": Challenges in delegating decisions to automated agents', in *Proceedings of the 2016 International Conference on Autonomous Agents #38 Multiagent Systems*, AAMAS '16, pp. 949–956, Richland, SC, (2016). International Foundation for Autonomous Agents and Multiagent Systems.

[9] Ingo Feinerer, 'Efficient large-scale configuration via integer linear programming', *Artif. Intell. Eng. Des. Anal. Manuf.*, **27**(1), 37–49, (January 2013).

[10] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[11] D. A. Ferrucci, 'Introduction to "This is Watson"', *IBM Journal of Research and Development*, **56**(3.4), 1:1–1:15, (May 2012).

[12] Chowdhury Gobinda G., 'Natural language processing', *Annual Review of Information Science and Technology*, **37**(1), 51–89.

[13] L. Hirschman and R. Gaizauskas, 'Natural language question answering: The view from here', *Nat. Lang. Eng.*, **7**(4), 275–300, (December 2001).

[14] M. Janota, *Do SAT Solvers Make Good Configurators?*, 191–195, Thiel, S., Pohl, K., Limerick, Ireland, 2008.

[15] Ulrich Junker, *Configuration.*, Chapter 24 of Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA, 2006.

[16] Stacy Marsella and Jonathan Gratch, 'Computationally modeling human emotion', *Commun. ACM*, **57**(12), 56–67, (November 2014).

[17] Francesca Rossi, Peter van Beek, and Toby Walsh, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*, Elsevier Science Inc., New York, NY, USA, 2006.

[18] G. Sierksma, *Linear and Integer Programming: Theory and Practice, Second Edition*, Advances in Applied Mathematics, Taylor & Francis, 2001.

[19] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis, 'Mastering the game of go with deep neural networks and tree search', *Nature*, **529**, 484–503, (2016).

[20] Peter Stone, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Sarit Kraus, Kevin Leyton-Brown, David Parkes, William Press, AnnaLee Saxenian, Julie Shah, Milind Tambe, and Astro Teller, 'Artificial intelligence and life in 2030', Technical report, Stanford University, Stanford, CA, (September 2016). One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel, Stanford University.

[21] Maite Taboada, 'Sentiment analysis: An overview from linguistics', *Annual Review of Linguistics*, **2**(1), 325–347, (2016).

[22] Mitchell M. Tseng and S. Jack Hu, *Mass Customization*, 836–843, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

# Group Decision Support for Requirements Management Processes

**R. Samer** and **M. Atas** and **A. Felfernig** and **M. Stettinger**
**Graz University of Technology, Graz, Austria**
**email: {rsamer, muesluem.atas, alexander.felfernig, martin.stettinger}@ist.tugraz.at**

**A. Falkner** and **G. Schenner**
**Siemens AG, Vienna, Austria**
**email: {andreas.a.falkner, gottfried.schenner}@siemens.com**

**Abstract.** Requests for proposal (RFP) trigger company-internal requirements management (RM) processes in order to assure that offers comply with a given set of customer requirements. As traditional RM approaches require a deep involvement of the requirements managers of a RM project especially when it comes to assigning suitable stakeholders to requirements, the quality of the decisions and the time effort for making correct decisions mainly depends on these experts. In this paper, we present a novel stakeholder assignment approach that reduces the overall involvement of these experts and also limits the uncertainty of overseeing suitable stakeholders at the same time. The assignment of responsible stakeholders is represented as a *group decision task* expressed in the form of a basic configuration problem. The outcome of such a task is a configuration which is represented in terms of an assignment of responsible stakeholders to corresponding requirements.

## 1 Introduction

Group-based configuration is an important application area of Artificial Intelligence [3, 4]. It aims to support a group of users in the configuration of complex products or services. In general, when interacting with group-based configurators, group members first articulate their preferences, then adapt inconsistent constraints, and finally, solutions are generated (i.e, reflecting the given configuration). In particular, when interacting with a configurator in the context of a typical requirements engineering task, each group member (i.e., stakeholder) has to evaluate each requirement according to different dimensions such as *priority, effort,* and *taken risk*. However, for the definition and evaluation of these requirements, first, suitable stakeholders have to be identified who are responsible for the development of these requirements. In addition, an early involvement of these stakeholders in the project is essential for the success of a project [5, 6, 13, 18]. This is because a low involvement of stakeholders in a project can lead to project failure. Project failures are often caused by missing or wrong assignments of stakeholders to requirements in early phases of the requirements engineering process [14]. Stakeholder recommendations can help to identify persons who are capable of providing a complete analysis and description of software requirements. Recommended stakeholders also need to bring deep knowledge about the corresponding item domain in order to provide precise evaluations of the requirements.

STAKENET [14] is an application that supports stakeholder identification on the basis of *social network analysis*. This approach builds a social network on the basis of a set of stakeholders. In this social network, stakeholders are represented by *nodes* and recommendations articulated by the stakeholders are represented by *links*. On the basis of such social networks, different *social network measures* are used for the prioritization of the stakeholders. One example of such a measure is *betweenness centrality* which measures the priority of a certain stakeholder *s* based on the ability of this user to play a role as a *broker* between separate groups of stakeholders. Castro-Herrera et al. [1] and Mobasher et al. [17] introduce a content-based recommendation approach where requirements are grouped by using different clustering techniques. Subsequently, stakeholders are recommended and assigned to these groups on the basis of content-based filtering. In this paper, a novel stakeholder assignment approach is introduced. The presented approach, acting as basic configuration service, lets voters evaluate stakeholders based on different criteria/dimensions and then aggregates their votes to derive possible configurations which are then recommended for the final stakeholder assignment decision to the requirements manager. In contrast to the aforementioned stakeholder recommendation approaches where the generated recommendations are directly suggested to the requirements manager, the content-based recommendation service presented in this paper *only* acts as a single *artificial voter* in addition to some human voters. Hence, the stakeholder recommendations (i.e., possible configurations) shown to the requirements manager are determined based on a combination of votes reflecting opinions of human voters as well as votes reflecting opinions of artificial voters.

The major contributions of this paper are the following. First, we analyze in detail a real-world scenario of a typical bid project. Second, we show an approach to identify relevant stakeholders for specific requirements and thus generate a global assignment of stakeholders to requirements. The remainder of this paper is organized as follows. In Section 2, we describe a typical application scenario of a bid project applied in an industrial context and provide a practical view of a traditional requirements management process commonly used for planning large industry projects. Additionally, a novel sophisticated approach is explained which further improves and ex-

tends the traditional approach by considering group decision support techniques. Section 3 discusses some potential issues and several factors this approach depends on. Subsequently, Section 4 explains the implementation of such an approach from a technical viewpoint. Finally, Section 5 concludes with a brief recap of this paper and presents some ideas for future work.

## 2 Application Scenario

Whenever an organizational unit of a large company (e.g., Siemens) decides to bid for a *Request for Proposal* (RFP), a new bid project for that proposal is initiated and the necessary stakeholders of the bid project are identified. RFPs for technical systems usually consist of a set of PDF or Microsoft Word documents which describe all requirements for the requested system covering technical, financial, legal, etc. aspects. Examples of stakeholders can be project managers, system architects, requirements managers, quality management departments, legal departments, engineering departments relevant for the bid, and potential external suppliers.

Within the context of a bid project, a requirements management (RM) process is initiated at the beginning. The purpose of this process is to assure that no requirement of the RFP has been overlooked. It involves the extraction of all the requirements contained in the RFP documents. The identified requirements must be assessed by the relevant stakeholders. This means that requirements concerning contracts must be assessed by the stakeholder(s) of the legal department, technical requirements must be assessed by the affected engineering department, etc. The assessment may involve statements about various criteria such as compliance, risks, approaches, etc. These statements are interpreted as *evaluation dimensions* in the remainder of this paper. At the end, each requirement of the RFP must have been assessed by at least one appropriate stakeholder.

### 2.1 Traditional RM Process

The *traditional requirements management* process can be best explained with an example. In the following, we describe a simplified example of a traditional RM process in a rail automation context based on a conventional RM tool such as *IBM DOORS*.

At the beginning, the requirements manager of the bid project creates a new project in the RM tool. After that, the necessary stakeholders for the current bid project are defined. In this context, stakeholders do not necessarily correspond to persons but correspond to roles which are uniquely identified with a unique string (called *Domain*). These string-based identifiers are unique within the organization. Furthermore, the RM tool supports the mapping of existing roles (i.e., domain identifiers) to concrete persons within the bid project. This way, responsible persons are assigned to roles based on their skills and domain knowledge.

Table 1 presents some examples of domain identifiers which occur in the context of rail automation. For such large bid projects usually more than 50 different domains are defined with the RM tool. However, in practice, most projects only use 20 different domains on average.

As a next step, the requirements manager imports all the relevant documents of the RFP into the project by using the RM tool. The RM tool automatically converts each paragraph of the documents into a (potential) requirement whilst the structure of the documents is preserved. The requirement manager then classifies the (potential) requirements in the project as either an actual requirement or as an arbitrary comment (called prose). In general, large infrastructure projects may contain more than 10,000 (potential) requirements.

| Domain | Stakeholder |
|---|---|
| PM | project manager |
| SA | system architect |
| RM | requirements manager |
| RAMS | reliability, availability, maintainability, and safety |
| S(ignal) | engineering department for railway signals |
| PS | engineering department for power supply |
| TVD | department for track vacancy detection |
| ETCS | department for European Train Control System |
| Test | quality management department |
| Supplier1 | external supplier, subcontractor |

**Table 1**: Examples of domain identifiers for rail automation

Each (actual) requirement must be assessed by at least one stakeholder. The requirements manager has to figure out which stakeholders are appropriate for which requirements and needs to assign them accordingly. However, other stakeholders may improve such initial assignments later during the assessment phase. The RM tool notifies all assigned stakeholders via e-mail to assess the requirements they are assigned to.

Table 2 shows an example of an initial assignment done by the requirements manager (RM). In this table, each row corresponds to a requirement and each column refers to a stakeholder. Each cell represents a single decision (of a stakeholder) for a stakeholder assignment (to a requirement). At the beginning, only the RM proposes assignments of potential stakeholders to requirements based on the manager's expertise and knowledge. For example, the assignment of $\{S, PM\}$ to the requirement $R5$ in the RM column indicates that $R5$ has been initially assigned to the *signal department* (S) and to the *project management department* (PM) by the requirements manager (RM). As only the RM makes assignments in this initialization phase, the values of all other columns remain empty (i.e., are filled with the "-" label) until the assessment phase.

| Req | RM | PM | RAMS | S(ignal) |
|---|---|---|---|---|
| R1 | {PM} | - | - | - |
| R2 | {PM} | - | - | - |
| R3 | {S} | - | - | - |
| R4 | {S} | - | - | - |
| R5 | {S, PM} | - | - | - |
| ... | | | | |

**Table 2**: Initial assignment of stakeholders to requirements done by requirements manager (RM). The dash symbol ("-") indicates that the other stakeholders have not made a decision yet.

Next, in the assessment phase, the affected stakeholders take a look at each of their assigned requirements in the RM tool and can either accept the requirement and assess it or they can veto the proposed assignment. Additionally, they can also propose an alternative stakeholder for the requirement or suggest (although rarely) an additional stakeholder for the requirement. For the remainder of this paper, this process is hereinafter referred to as *assignment feedback*. After that, the requirements manager can either accept the veto and assign the requirement to a different stakeholder or decline the veto and reassign the stakeholder to the requirement.

Table 3 shows an intermediate state during the assignment phase which demonstrates examples of *assignment feedback* given by the stakeholders $PM$ and $S(ignal)$:

| Req | RM | PM | RAMS | S(ignal) |
|---|---|---|---|---|
| R1 | {PM} | {PM} | - | - |
| R2 | {PM} | {RAMS} | - | - |
| R3 | {S} | - | - | {S, RAMS} |
| R4 | {S} | - | - | {} |
| R5 | {S, PM} | {S, PM} | - | {S, PM} |
| ... | | | | |

**Table 3**: State of assignment during assessment phase

- Requirement $R1$ has been accepted by $PM$
- Requirement $R2$ has been vetoed by $PM$ and $RAMS$ has been proposed by $PM$ as alternative stakeholder
- Requirement $R3$ has been accepted by $S(ignal)$, but $RAMS$ has been proposed by $S(ignal)$ as an additional stakeholder
- Requirement $R4$ has been vetoed by $S(ignal)$
- Requirement $R5$ has been accepted by all proposed stakeholders

It is important to point out the fact that in the traditional scenario, it is always the main responsibility of the requirements manager to resolve potential conflicts. Typically, this usually involves some personal discussions with the involved stakeholders and some final decisions made by the requirement manager. These final decisions then assure a consistent assignment of all requirements to responsible stakeholders. Table 4 presents such a final state where all conflicts have been resolved.

| Req | RM | PM | RAMS | S(ignal) |
|---|---|---|---|---|
| R1 | {PM} | {PM} | - | - |
| R2 | {RAMS} | {RAMS} | {RAMS} | - |
| R3 | {S, RAMS} | - | {S,RAMS} | {S, RAMS} |
| R4 | {S} | {S} | - | {S} |
| R5 | {S, PM} | {S, PM} | - | {S, PM} |
| ... | | | | |

**Table 4**: Final state after assessment phase. Consistent assignment of stakeholders to requirements.

The requirements manager periodically reminds the assigned stakeholders about their unassessed requirements. This process is repeated until all requirements have been assessed and the assessment phase is finished. Thus, the assignment of stakeholders can be considered as a manual configuration process. The outcome of this process is a configuration in terms of a consistent assignment of stakeholders to requirements they are responsible for. In our current implementation, the overall goal is to achieve *consensus* regarding the stakeholder assignment. Future versions of our system include further constraints that have to be taken into account in task allocation tasks as discussed in this paper.

## 2.2 RM Process with Group Decision Support

The main idea of our novel requirements management approach is to introduce additional stakeholder votes made by *artificial stakeholders* (called *bots*). Additionally, the bots automatically propose stakeholders in the initial phase of the RM process. Furthermore, an intelligent *group decision service* is included in the RM tool to automatically aggregate all votes given by human stakeholders as well as artificial stakeholders. On a technical level, such a *group decision service* represents a *group recommender system* which generates recommendations based on aggregated votes given by group members of a group (i.e., the stakeholders) [2]. Basically, there exist different strategies on how to aggregate votes of group members [8] such as *majority*, *average*, *least-misery*, etc. In addition, more sophisticated aggregation functions exist - for further information regarding preference aggregation functions we refer to [2, 15]. To limit the scope of this paper, we assume that the group decision service is a simple group recommender using basic aggregation strategies.

The votes of the artificial stakeholders (i.e., bots) are generated by using appropriate content-based recommendation algorithms (see Section 4). This way, the group decision service allows to replace the traditional mainly manual stakeholder assignment process (see Section 2.1) with a semi-automatic process. As a key difference to the traditional approach, the group decision service automatically aggregates the decisions of all voters and thereby allows the smart incorporation of additional (automatic) voters, i.e., intelligent recommendation services for stakeholder assignments. From an abstract point of view, the process can be interpreted as a basic configuration process. Like in the traditional RM process (see Section 2.1), the outcome of this process represents a consistent assignment of stakeholders to requirements they are responsible for.

Table 5 illustrates a possible initial state in the presence of a group decision service ($GDS$) and a stakeholder assignment recommendation service (denoted as $RS1$). In sharp contrast to the assignments made by other stakeholders, the recommendation service does not provide a binary decision for every stakeholder but a confidence value which lies in the range between 1 and 10, whereby a higher number corresponds to more confidence and a lower number corresponds to a lower level of confidence.

The column for the $GDS$ shows the result of the group decision service for each requirement, i.e., the aggregated decision of all voters (including humans and bots/algorithms). Note that a clear benefit of the group decision service is that some requirements can already be assessed by the assigned stakeholders, even though they have not yet been proposed/assigned by the requirements manager. In other words, stakeholders are automatically proposed by the bots/algorithms based on their skills in the *initial phase* and can already evaluate their assignment to the requirements. Hence, much assignment effort is taken away in the initial phase from the time-pressured requirements managers and the initial phase can be significantly speeded up. Moreover, it is necessary to point out that the stakeholders $GDS$ (perform aggregation) and $RM$ (perform final decision) can be considered to have a special role in this evaluation process, whereas all other stakeholders only occur as voters in the process. Consequently, the major responsibility/task of a $RM$ in this process is to review the decision suggested by the $GDS$ and to perform the final decision about the assignment of the stakeholders to the requirements.

# 3 Potential Issues of Group Decision Support

The exact behavior of the new system presented in Section 2.2 will depend on various factors. Examples of such factors include the *aggregation strategy* used by the group decision service to aggregate the votes (e.g., majority, average, etc.), the individual weight of the voters (e.g., "deciders"/experts count higher than normal stakeholders), and the confidence/trust users have in different recommendation algorithms.

Furthermore, the question arises how conflicting decisions (for example, stakeholder A assigns stakeholder B and B assigns A) can be resolved or supportive advice to manually resolve such conflicts can be given to the voters by the system. Also, inconsistencies and contradictions may occur in the evaluation of stakeholders between the voters. These voters can be other stakeholders and *artificial stakeholders*. In particular, for artificial stakeholders textual explanations

| Req | GDS | RS1 | RM | PM | RAMS | S(ignal) |
|---|---|---|---|---|---|---|
| R1 | {PM} | {PM:9} | {PM} | - | - | - |
| R2 | {RAMS} | {RAMS:8, PM:5} | - | - | - | - |
| R3 | {S} | {S:8, RAMS:6} | - | - | - | - |
| R4 | {S} | {S:5} | - | - | - | - |
| R5 | {S} | {S:6} | {S,PM} | - | - | - |
| ... | | | | | | |

**Table 5**: State of assignment with group decision service (GDS) and stakeholder recommendation service (RS1). The recommendation service provides a confidence value which lies in the range between 1 and 10.

can be presented to the group of voters being in conflict. Such textual explanations can then express the concrete reason and arguments for the votes provided by the artificial stakeholders.

Moreover, the prediction quality (i.e., performance) of the *artificial stakeholders* (i.e., the recommender systems) plays a major role in the process. In particular, the generated recommendations should be evaluated and examined with respect to completeness. In terms of common information retrieval measures (such as precision and recall), this would, for example, mean that more emphasis should be given to the recall of the results rather than the precision achieved by the recommender. In addition to that, an appropriate recommendation algorithm should also be capable of giving negative indication by telling the RM which stakeholders are definitely not suitable to be assigned to a requirement at all. Such a negative indication can be shown as, e.g., RAMS:0. Finally, another important aspect would be to take the availability of stakeholders into account before they get finally assigned to a requirement. This adds another complexity dimension to the underlying basic configuration problem.

## 4 Group Decision Support for Bidding Processes

In this section, a slightly modified version of the aforementioned RM process based on *Group Decision Support* (see Section 2.2) is described. The description explains the technical implementation of this process provided by the requirements engineering platform OPENREQ MVP[1] which is developed within the scope of the OpenReq EU Horizon 2020 research project. At the current stage, the implementation is already in use, however, still ongoing and ready to be further enriched with additional features. The remainder of this section describes the current status of the existing implementation.

In the initial phase, the requirements manager (RM) is asked by the system to propose suitable stakeholders for each requirement. As already described in Section 2.2, a content-based recommender system (RS1) helps the RM to find stakeholders based on keywords extracted from former requirements those stakeholders have solved. Thereby, on an abstract level, the automated *stakeholder-recommendation* algorithm (of RS1) can be interpreted as a *text classification* task [7] where the recommendation algorithm exploits several *Natural Language Processing* [19, 20] techniques in order to correctly classify stakeholders suitable for a given requirement.

The algorithm automatically extracts relevant keywords from the title and description text of all former requirements which a stakeholder was assigned to, in order to build a user profile for the respective stakeholder. First, the title and description text is cleaned by removing special characters (such as ".", ",", ";", "#", etc.). Next, the text is split into tokens (which, basically, represent the words in the text) and *stop words* such as prepositions (e.g., "in", "on", "at", etc.) or articles (e.g., "the", "a", "an") are removed. After applying *Part-of-speech tagging*, tokens/words of classes (such as verbs, adjectives, or numbers) that are most probably irrelevant to be used as keywords are removed. Finally, the remaining tokens of each former requirement (which was assigned to the stakeholder) are merged together into a single user profile.

By applying the same procedure to new requirements, keywords for new requirements are extracted as well. Given the keywords of a new requirement and the user profiles of the individual stakeholders, a similarity between a new requirement and a stakeholder is calculated for every stakeholder provided that the stakeholder has been assigned to an (already completed) requirement in the past. Formula 1 shows the Dice coefficient formula [9] which is a variation of the Jaccard coefficient and used to compute the similarity between a stakeholder and a requirement. The similarity is measured by comparing the overlap of the keywords of the stakeholder's user profile (denoted as $U_a$) and the relevant keywords of the respective requirement (denoted as $r_x$) with the total number of keywords appearing in $U_a$ as well as $r_x$.

$$sim(U_a, r_x) = \frac{2 * |keywords(U_a) \cap keywords(r_x)|}{|keywords(U_a)| + |keywords(r_x)|} \quad (1)$$

Stakeholders who are most similar to a given requirement are suggested by the content-based recommender to the RM. This way, the initial phase can be speeded up and the chance of overseeing suitable stakeholders for requirements at this early stage of the process, is decreased. In the next step, the OPENREQ MVP system shows a list of the initially assigned stakeholders for each requirement. Stakeholders who are assigned to a requirement can either accept or reject their assignment. In addition, the assignments of the stakeholders for the requirement can be evaluated by all stakeholders.

This evaluation of a stakeholder-assignment is done based on the criteria *Appropriateness* and *Availability* (see Figure 1). Both criteria are interpreted as *evaluation dimensions* and stakeholders are evaluated based on both dimensions. Furthermore, an assigned stakeholder can also propose the assignment of further stakeholders to the requirement. These newly assigned stakeholders can then be evaluated again. After a new vote has been given, the *group decision service* (GDS) is triggered to compute a utility value for the rated stakeholder. Formula 2 shows the calculation of the utility value of an evaluated stakeholder $s$, whereas $D$ is the set containing both dimensions, i.e., $D = \{Appropriateness, Availability\}$.

$$utility(s, r) = \frac{\sum_{t \in T} \frac{\sum_{d \in D} eval(s,r,d,t) \cdot weight(d)}{\sum_{d \in D} weight(d)}}{|T|} \quad (2)$$

The formula describes the stakeholder $s$ to be voted by other stakeholders, whereby $T$ represents the set of stakeholders $t \in T$ who evaluated $s$. More formally expressed, $T$ is a set which contains the

[1] OpenReq MVP: `http://openreq.ist.tugraz.at`

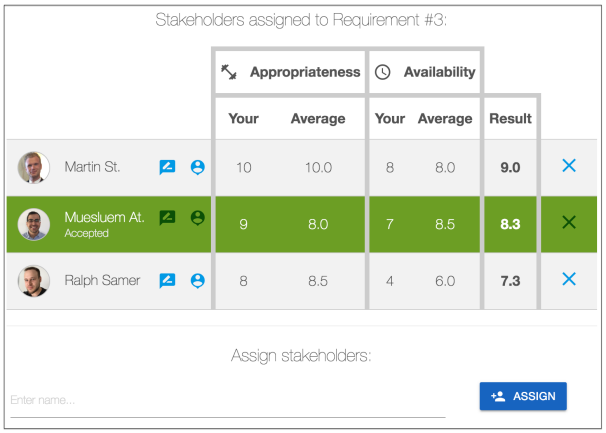


**Figure 1**: Evaluation of stakeholders in OPENREQ MVP. Each stakeholder-assignment is evaluated by two evaluation dimensions (appropriateness and availability). The utility value of an evaluated stakeholder is calculated by using Formula 2.

stakeholders (including $s$) who evaluated stakeholder $s$, i.e., $T \subseteq S$. Furthermore, the OPENREQ MVP platform allows the requirements manager to define different importance levels for both dimensions. In Formula 2, the importance of a dimension $d \in D$ is expressed by the function $weight(d)$. Moreover, $eval(s, r, d, t)$ refers to the dimension-specific rating given by stakeholder $t$ for stakeholder $s$ for the requirement $r$. Finally, the result of $utility(s, r)$ represents the aggregated utility of a stakeholder $s$ for requirement $r$.

Once all assignments have been evaluated by a sufficient number of stakeholders, a stable state of the assignment utilities is achieved. The utility values are then used as main feedback source for the requirements manager to make the final decision about which stakeholder(s) should be assigned to the requirement.

## 5 Conclusion and Future Work

*Conclusion.* In this paper, we discussed common application scenarios of requirements engineering in the context of industry projects. These scenarios range from traditional requirements management processes where the assignment process of stakeholders is solely controlled by the requirements manager, to more sophisticated automated approaches where the involvement of the requirements manager is reduced to a minimum. The latter represents a basic configuration service which includes *artificial stakeholders* as additional voters and a *group decision support system* as a vote aggregation component in the evaluation of stakeholder assignments to requirements. On the basis of this scenario we showed how these two components can be applied in order to improve the requirements management process such that the overall effort and the chance of overseeing stakeholders suitable for requirements can be reduced for the time-pressured requirements managers.

*Future Work.* As bidding processes can be seen as repetitive processes, mechanisms which are capable of learning stakeholder weights and taking individual expertise levels of stakeholders into account can be considered as potential ideas regarding future work. Moreover, the set of existing evaluation dimensions can be further extended such that more fine-grained control is given to the evaluation process as well as to the *group decision service*. Additionally, the concept of *liquid democracy* can be integrated into the evaluation process [10]. This way, stakeholders who do not have sufficient knowledge concerning the details of a requirement can easily delegate their votes to more well-informed and experienced experts.

With respect to conflicting decisions (see Section 3), future work should also include mechanisms to automatically resolve such conflicts or mechanisms which provide supportive advice to the voters, showing how they can manually resolve such conflicts. Furthermore, the configuration approach can be enriched with further constraints taking resource management aspects of stakeholders into consideration, in order to optimize the overall allocation of human resources in release planning.

Finally, there is also still plenty of room for improvement regarding the extraction of keywords used by the discussed content-based recommender system (i.e., *artificial stakeholder*). For example, a more descriptive and characteristic representation of the keywords can be obtained by using more sophisticated content-based approaches such as *Latent Semantic Analysis* (LSA) [11, 16] or *word2vec* algorithms [12, 16].

## Acknowledgment

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OPENREQ (732463).

## REFERENCES

[1] Carlos Castro-Herrera, Chuan Duan, Jane Cleland-Huang, and Bamshad Mobasher, 'Using data mining and recommender systems to facilitate large-scale, open, and inclusive requirements elicitation processes', 165–168, (09 2008).
[2] A. Felfernig, L. Boratto, M. Stettinger, and M. Tkalcic, *Group Recommender Systems – An Introduction*, Springer, 2018.
[3] Alexander Felfernig, M Atas, T Tran, and Martin Stettinger, 'Towards group-based configuration', in *International Workshop on Configuration 2016 (ConfWS16)*, pp. 69–72, (2016).
[4] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.
[5] Alexander Felfernig, Martin Stettinger, Andreas Falkner, Muesluem Atas, Xavier Franch, and Christina Palomares, 'Openreq: Recommender systems in requirements engineering', pp. 1–4, (10 2017).
[6] Hubert F. Hofmann and Franz Lehner, 'Requirements engineering as a success factor in software projects', *IEEE software*, **18**(4), 58, (2001).
[7] Emmanouil Ikonomakis, Sotiris Kotsiantis, and V Tampakas, 'Text classification using machine learning techniques', **4**, 966–974, (08 2005).
[8] Anthony Jameson and Barry Smyth, 'The adaptive web', chapter Recommendation to Groups, 596–627, Springer-Verlag, Berlin, Heidelberg, (2007).
[9] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich, *Recommender Systems: An Introduction*, Cambridge University Press, New York, NY, USA, 1st edn., 2010.
[10] Anson Kahng, Simon Mackenzie, and Ariel D. Procaccia, 'Liquid democracy: An algorithmic perspective', in *AAAI*, (2018).
[11] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham, 'An introduction to latent semantic analysis', (1998).
[12] Jey Han Lau and Timothy Baldwin, 'An empirical evaluation of doc2vec with practical insights into document embedding generation', *CoRR*, **abs/1607.05368**, (2016).
[13] Dean Leffingwell, 'Calculating your return on investment from more effective requirements management', *American Programmer*, **10**(4), 13–16, (1997).
[14] S.L. Lim, D. Quercia, and A. Finkelstein, 'Stakenet: Using social networks to analyse the stakeholders of large-scale software projects', in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pp. 295–304, New York, NY, USA, (2010). ACM.

[15] J. Masthoff, 'Group recommender systems', *Recommender Systems Handbook*, 677–702, (2011).
[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, 'Distributed representations of words and phrases and their compositionality', in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 3111–3119, USA, (2013). Curran Associates Inc.
[17] B. Mobasher and J.Clehand-Huang, 'Recommender systems in requirement engineering', 81–89, (2011).
[18] Bamshad Mobasher and Jane Cleland-Huang, 'Recommender systems in requirements engineering', *AI magazine*, **32**(3), 81–89, (2011).
[19] Kevin Ryan, 'The role of natural language in requirements engineering', in *[1993] Proceedings of the IEEE International Symposium on Requirements Engineering*, pp. 240–242, (Jan 1993).
[20] J. Winkler and A. Vogelsang, 'Automatic classification of requirements based on convolutional neural networks', in *2016 IEEE 24th International Requirements Engineering Conference Workshops(REW)*, volume 00, pp. 39–45, (Sept. 2016).

# Chatbot-based Tourist Recommendations Using Model-based Reasoning

**Iulia Nica** and **Oliver A. Tazl** and **Franz Wotawa**[1]

**Abstract.** Chatbots have gained increasing importance for research and practice with a lot of applications available today including Amazon's Alexa or Apple's Siri. In this paper, we present the underlying methods and technologies behind a Chatbot for e-tourism that allows people textually communicate with the purpose of booking hotels, planning trips, and asking for interesting sights worth being visit. In particular, we show how model-based reasoning can be used for enhancing user experience during a chat, e.g., in cases where too many possible selections are available or where user preferences are too restricted causing inconsistencies and as a consequence not possible answers to be provided. Besides the underlying foundations, we provide a use case from the intended tourism domain to show how such a model-based chatbot effectively can be used in practice.

## 1 Introduction

Communicating with systems based on natural language is very much appealing and of growing interest and importance also for industry. See for example [1, 2] for predictions about the rise of the chatbot market in the future. Natural language interfaces (NLI) offer a lot of new possibilities for humans to interact and collaborate with users [8]. Chatbots are a form of artificial intelligence system that allows a human-computer interaction in a natural language form. They could be based on rule sets or neural networks in order to decide the correct answer to the user's request. Chatbots are not restricted to certain application domains. They are flexible enough to be used in many different application scenarios and domains including systems for tourists recommending sights, hotels, or even complete travel plans. Often chatbots rely on pre-specified patterns that trigger the chatbot's behavior, e.g., see [25], restricting its space of interaction with users.

In this paper, we focus on improving adaptivity of chatbots in the context of recommender systems, where we have identified two issues that arise during and human-computer interaction session. In order to make a recommendation, the chatbot has to interact with the user in order to find out preferences and wishes in order to make an appropriate recommendation. In case of too little preference information, the *first issue* is, that a chatbot may *not be able to restrict the number of recommendations* to be provided to the user. Selecting a particular recommendation, e.g., the first one in a list of 1,000 elements might not be the best idea. It might also be not possible to find a general applicable function that returns the best solution for the current user. Hence, there is a need to further restrict the search space, which can be provided by asking the user about further preferences that allow to restrict the search space in an optimal way.

[1] TU Graz, Institute for Software Technology, Inffeldgasse 16b/2, A-8010 Graz, Austria, email: {inica, oliver.tazl, wotawa}@ist.tugraz.at

The *second issue* that may arise is the impossibility of providing even a single recommendation because of *inconsistent or too restrictive preferences provided by the user*. In this case, it is necessary, to provide feedback to the user and ask for removing preferences or for ranking preferences accordingly to their importance. In this paper, we focus on these two issues and provide a solution for both. For the *first issue*, we propose the use of Entropies for selecting new preferences. For the *second issue*, we suggest using model-based diagnosis for identifying the causes of inconsistencies. In addition, we put both parts together in a single recommendation algorithm for improving user experience when interacting with a chatbot.

The main contributions of this paper can be summarized as the follows.

1. An algorithm that is based on the ideas of model-based diagnosis and Shannon's information entropy to solve recommendation problems.
2. An iterative approach to the algorithm realized with a natural language interface using a chatbot.

The remainder of this paper is organized as follows: In the next section we introduce an example domain and give an overview of our algorithmic approach. Afterwards, we get into more details regarding the designed algorithms. Finally, we discuss related research and conclude the paper.

## 2 Chatbot for Tourism – A Case Study

In this section, we discuss the use of chatbots for recommendation in the area of tourism where we introduce a typical conversation between a tourist and a chatbot. This conversion presents a use case of a tourism chatbot and serves as motivating example throughout this paper. We depict the whole conversation in Figure 1, which illustrates the recommendation process for hotels in a specific area. There, after selecting the sight of interest, i.e., the Branderburg gate, the tourist requests a low-priced hotel near this sight. The answer of the chatbot from Line 14 in Figure 1 introduce a first challenging situation of every recommender, i.e., an unsuccessful user query. An intelligent recommender should be able to deal with such a situation and provide a list of items which fulfill as many requirements as possible, and ideally explain the cause of trouble. Therefore, the next step for the recommender should be to automatically identify the minimal set of inconsistent requirements provided by the user. Furthermore, a flexible, interactive query-based approach should allow the user to choose from the identified inconsistent requirements, and as a consequence enhancing the recommendation. The second corner situation, which is illustrated in Line 18, deals with the case where we have too many hotels matching the customer requirements. In this situation,

```
1  Bot: Hello John, how can I help you today?
2  John: I would like to plan a trip.
3  Bot: Great! Are you interested in a specific region
4         or would you like to plan the trip by activities?
5  John: I am interested in a specific region.
6  Bot: Please click on map to pick the point of interest
7         or type in the place that you are interested in.
8   (John clicks a point on the map near the city Berlin)
9  Bot: OK, I will search for relevant sights near Berlin.
10 (Shows a list of relevant sights in Berlin.)
11 John: Show me the Brandenburger Tor.
12 (The bot shows several information about the Brandenburger Tor.)
13 John: Looks nice! Find a low-priced hotel near this sight.
14 Bot: I am sorry, but there are no hotels of this category near
15      Brandenburger Tor. Which requirement is not that important
16      to you? Price or distance?
17 John: Distance.
18 Bot: OK! There are 9 hotels available for you. You should
19      incorporate another filter. How about parking? Will
20      you have a car?
21 John: No, I do not need a parking facility.
22 Bot: OK! Now there are 4 hotels left. Here is the list: Hotel 7,
23      Hotel 10, Hotel 15, Hotel 18.
24 John: Cool! Is Hotel 15 available from 19.08.2018 to 25.08.2018
25       for two people?
26 Bot: Yes! Should I book it?
27 John: Yes.
28 Bot: Hotel booked!
```

**Figure 1**: An Example Conversation

the system should be able to generate a proposal for the user, suggesting which hotel attribute or feature should be further constrained to narrow down the number of relevant hotels. Here the chatbot could randomly offer alternative features to choose from, or even better, it could optimize the searching process by making use of an algorithm to determine the attribute, which if constrained, leads to the largest information gain. Note that we rely on a pure knowledge-based approach, i.e., we assume that no other filtering like collaborative or content-based is available.

| name | distance | category | parking | price |
|---|---|---|---|---|
| Hotel 1 | med | 5-stars | false | med |
| Hotel 2 | med | 5-stars | true | high |
| Hotel 3 | med | 3-stars | true | low |
| Hotel 4 | long | 3-stars | true | low |
| Hotel 5 | med | 5-stars | false | high |
| Hotel 6 | short | 4-stars | true | high |
| Hotel 7 | long | 2-stars | false | low |
| Hotel 8 | med | 5-stars | false | high |
| Hotel 9 | long | 3-stars | true | low |
| Hotel 10 | long | 2-stars | false | low |
| Hotel 11 | short | 4-stars | true | high |
| Hotel 12 | short | 3-stars | false | med |
| Hotel 13 | med | 5-stars | true | high |
| Hotel 14 | short | 3-stars | false | med |
| Hotel 15 | long | 3-stars | false | low |
| Hotel 16 | long | 3-stars | true | med |
| Hotel 17 | med | 3-stars | true | low |
| Hotel 18 | med | 2-stars | false | low |
| Hotel 19 | long | 2-stars | true | low |
| Hotel 20 | long | 4-stars | true | med |

**Table 1**: Item set for $type = hotel$

In the following, we further discuss the details of how to overcome the considered issues that arise during the recommendation process, i.e., not being able to provide a recommendation due to inconsistencies, and not being able to reduce the number of selection given the current requirements. In this discussion we focus only on those parts that are important for the case study for the sake of clarity. Therefore, we further consider only the items of type $hotel$ to be part of the knowledge base, which we depict in Table 1. There we further assume that each $hotel$ possesses a simplified set of attributes, containing the $name$ of the hotel, its $price$, defining the price range (low, medium, or high), its $category$ with the domain {2-stars, 3-stars, 4-stars, 5-stars}, the availability of $parking$ space being either true or false, and $distance$ (short, medium, or long). Note that $distance$ is a special attribute, as it represents the distance to the starting point introduced by the user in the current recommendation session and thus it has to be recalculated on demand and not actually stored in the knowledge base.

It is easy to see that the requirements specified by the customer in Line 13 cannot be satisfied by the items from Table 1:

$$R_1 = \{r_1 : distance = short; r_2 : price = low\},$$

as there is no low-priced hotel within $distance = short$ in the given assortment. Hence, we are interested in identifying that minimal set of requirements that when changed, lead to a recommendation for the customer. In our example, the situation is simple. The recommendation system determines that either $r1$ or $r2$ have to be relaxed (in the sense that the chosen requirement will not be further taken into consideration when computing the recommendation). Still, in more complex scenarios, when the user query implies more requirements, the solution is not so straightforward. For instance, if the query was: " Find a low-priced, 4-stars hotel near this sight.", then we would have to deal with the following user requirements:

$$R_2 = \{r_1 : distance = short; r_2 : price = low; \\ r_3 : category = 4 - stars\}.$$

There choosing $r_1$ alone as inconsistent requirement would not solve the problem, because, as one can see in Table 1, we still have no items that satisfy both $r_2$ and $r_3$. Hence, in order to automatically identify the minimal set of inconsistent requirements, we would have to make use of logical reasoning methods that are able to determine causes for inconsistencies, e.g., consistency-based reasoning, where we have to describe the inconsistent requirements problem as a diagnosis problem. The idea is not new and state-of-the-art knowledge-based approaches like [6, 7, 14, 21] compute minimal sets of faulty requirements, which should be changed in order to find a solution. In this paper, we take the idea and transfer it to the domain of chatbots. In addition, we do not need to come up with conflicts for computing diagnoses but instead compute inconsistent requirements directly from the given formalized knowledge. Furthermore, the idea of personalized repairs is covered by asking the user directly which requirements he or she prefers. We discuss the recommendation algorithm in detail in Section 3.

In the following, we now discuss the other particular situation, where a recommender would have to deliver too many items matching the customer's requirements. In order to determine which attribute selection is the best one for accelerating the searching process, we suggest computing the entropy of the items' attributes at the

| name | distance | category | parking | price |
|---|---|---|---|---|
| Hotel 3 | med | 3-stars | true | low |
| Hotel 4 | long | 3-stars | true | low |
| Hotel 7 | long | 2-stars | false | low |
| Hotel 9 | long | 3-stars | true | low |
| Hotel 10 | long | 2-stars | false | low |
| Hotel 15 | long | 3-stars | false | low |
| Hotel 17 | med | 3-stars | true | low |
| Hotel 18 | med | 2-stars | false | low |
| Hotel 19 | long | 2-stars | true | low |

**Table 2**: Solution list for the user query $distance = \{med, long\}, price = low$

first place. Using entropies for finding the best next selection in order to accelerate the overall search process is not new. For example, De Kleer and Williams [3] introduced a measurement selection algorithm for obtaining the next best measurement in order to reduce the diagnosis search space. In our case, we have a similar situation and adapt using Entropies for our purpose. Further not that entropy is a measure commonly used in decision and information theory to quantify choice and uncertainty. For more details on Shannon's information entropy, we refer the interested reader to [24].

Let us consider our case study. In this example, the user query $R = \{distance = \{med, long\}; price = low\}$ leads to the solution list given in Table 2. There we have a set of 9 hotels with the attributes $distance, category, parking$, and $price$. In order to further reduce the number of hotels provided by the recommender, we have to identify the next attribute that should be further constraint by the user. In case of entropy used for selection, we have to compute the entropy for each attribute first. For more details on Shannon's information entropy we refer the reader to [24]. For computing the entropy of an attribute $X$, we make use of the following formula from [24] where $P(x_i)$ is the probability that attribute $X$ takes value $x_i$:

$$H(X) = -\sum_i P(x_i) \log P(x_i) \tag{1}$$

Entropy has several interesting properties. Among them, as Shannon mentions in [24], $H = 0$ if and only if all the $P(x_i)$ but one are zero. Thus only when we are certain of the outcome does $H$ vanish, otherwise $H$ is positive. In the other extreme case, for a given $n$, $H$ is a maximum and equal to $\log n$ when all the $P(x_i)$ are equal to $1/n$.

Let us now make use of entropies for selecting the next best attribute. Hence, we compute the attributes' entropies as follows:

$$\begin{aligned} H(distance) &= -P(med)\log P(med) - P(long)\log P(long) \\ &= -1/3\log(1/3) - 2/3\log(2/3) \\ &= 0.92 \end{aligned}$$

$$\begin{aligned} H(category) &= -P(3s)\log P(3s) - P(2s)\log P(2s) \\ &= -5/9\log(5/9) - 4/9\log(4/9) \\ &= 0.99 \end{aligned}$$

$$\begin{aligned} H(parking) &= -P(t)\log P(t) - P(f)\log P(f) \\ &= -5/9\log(5/9) - 4/9\log(4/9) \\ &= 0.99 \end{aligned}$$

$$\begin{aligned} H(price) &= -P(low)\log P(low) \\ &= 9/9\log(9/9) \\ &= 0 \end{aligned}$$

From these figures we see that we obtain the maximum entropy for the attributes $category$ and $parking$, whereas the minimum entropy is computed for $price$. In order to make the best choice, the recommendation system offers the attribute with the largest entropy value, i.e., $category$ or $parking$ in our case, to the user and asks him or her to further constrain it via selecting a certain attribute value. If the number of the remaining recommendations still exceeds a predefined maximum number of recommendations, the described solution reduction process based on entropy continues with the second best entropy attribute as already described above. In the next section, we describe an algorithm implementing this process in more detail and also integrate it within a whole recommendation process loop.

## 3 EntRecom Algorithm

Before stating our recommendation algorithm, which is based on a diagnosis algorithm that is close to ConDiag [20], and on a method that applies Shannon's information entropy [24] for the attributes, we introduce and discuss basic definitions. We first formalize the inconsistent requirements problem, by exploiting the concepts of Model-Based Diagnosis (MBD) [3, 22] and constraint solving [4].

The inconsistent requirements problem requires information on the item catalog (i.e., the knowledge-base of the recommendation system) and the current customer's requirements. Note that the knowledge-base of the recommender may be consistent with the customer's requirements (i.e., the customer's query) and an appropriate number of recommendations can be offered. In this case, the recommendation system shows the recommendations to the customer and no further algorithms have to be applied. Otherwise, if no solutions to the recommendation problem were found, then the minimal set of requirements, which determined the inconsistency with the knowledge base, have to be identified and consequently offered to the user as explanation for not finding any recommendation. The user can in this case adapt the requirement(s) (relax it/them). Here we borrow the idea from MBD and introduce abnormal modes for the given requirements, i.e., we use $Ab$ predicates stating whether a requirement $i$ is should be assumed valid ($\neg Ab_i$) or not ($Ab_i$) in a particular context. The $Ab$ values for the requirements are set by the model-based diagnosis algorithm so that the assumptions together with the requirements and the knowledge-base are consistent. In the following, we define the inconsistent requirements problem and its solutions.

We start stating the inconsistent requirements problem:

**Definition 1 (Inconsistent Requirements Problem)** *Given a tuple $(KB, REQ)$ where $KB$ denotes the knowledge base of the recommender system, i.e., the item catalog, and $REQ$ denotes the customer requirements. The Inconsistent Requirements Problem arises*

*when $KB$ together with $REQ$ is inconsistent. In this case we are interested in identifying those requirements that are responsible for the inconsistency.*

For our example introduced in Section 2, there is a knowledge base $KB$ capturing the rows of the Table 1. This can be formalized as follows: $(name = Hotel\_1 \wedge distance = med \wedge category = 5 - stars \wedge parking = false \wedge price = med) \vee (name = Hotel\_2 \wedge distance = med \wedge category = 5 - stars \wedge parking = true \wedge price = high) \vee \ldots$. We have formalized knowledge stating equations, i.e., saying that $distance$ cannot be $short$ and $med$ at the same time, i.e., $distance = short \wedge distance = med \rightarrow \bot$. In addition, there are two requirements $REQ = \{R1, R2\}$, and for each requirement a variable $Ab_{R1}, Ab_{R2}$ stating whether the requirement should be considered or not. The requirements $R1$, and $R2$ themselves can be defined using the following logical representation $Ab_{R1} = 0 \rightarrow (distance = short)$, and $Ab_{R2} = 0 \rightarrow (price = low)$ respectively. Obviously, when assuming all $Ab_{Ri}$ (for $i = 1, 2$) to be 0, we obtain an inconsistency because there is no hotel matching the requirements. Therefore, an explanation for such inconsistencies is needed.

A solution or explanation to the inconsistent requirements problem can be easily formalized using the analogy with the definition of diagnosis from Reiter [22]. We first introduce a modified representation of $(KB, REQ)$ comprising $(KB_D, REQ)$ where $KB_D$ comprises $KB$ together with rules of the form $Ab_R$ for each requirement $R$ in $REQ$. The solution to the Inconsistent Requirements Problem can now be defined using the modified representation as follows:

**Definition 2 (Inconsistent Requirements)** *Given a modified recommendation model $(KB_D, REQ)$. A subset $\Gamma \subseteq REQ$ is a valid set of inconsistent requirements iff $KB_D \cup \{\neg Ab_R | R \in REQ \setminus \Gamma\} \cup \{Ab_R | R \in \Gamma\}$ is satisfiable.*

A set of inconsistent requirements $\Gamma$ is minimal iff no other set of inconsistent requirements $\Gamma' \subset \Gamma$ exists. A set of inconsistent requirements $\Gamma$ is minimal with respect to cardinality iff no other set of inconsistent requirements $\Gamma'$ with $|\Gamma'| < |\Gamma|$ exists. From here on we assume minimal cardinality sets when using the term minimal sets.

For our example, inconsistent requirements are $\{R1\}$ and $\{R2\}$. In both cases there are hotels available and we do not obtain an inconsistency any more. In the following, we describe the algorithm for providing recommendations in the context of chatbots.

Algorithm 1 **EntRecom** takes a knowledge base, a set of customer requirements, and the maximum number of recommendations, and computes all recommendations. Algorithm 1 is an iterative algorithm that starts with deriving a constraint model $CM$ from the knowledge base $KB$ and the customer requirements $REQ$. Such a constraint representation captures the semantics of the provided knowledge base and requirements. Following the ideas presented in [20], we use a constraint solver both to directly compute the recommendations and to determine the inconsistent requirements. Still, in contrast to ConDiag, which guarantees to compute all the minimal diagnoses up to a predefined cardinality, we are interested here only in the minimal cardinality diagnosis, that in our case translates to the minimal set of inconsistent requirements.

Therefore, in Step 2, we check the consistency of our model by calling **CSolver**, a constraint solver taking the set of constraints $CM$

**Algorithm 1 EntRecom**$(KB_D, REQ, n)$

**Input:** A modified knowledge base $KB_D$, a set of customer requirements $REQ$ and the maximum number of recommendations $n$
**Output:** All recommendations $S$

1: Generate the constraint model $CM$ from $KB_D$ and $REQ$
2: Call $\mathbf{CSolver}(CM)$ to check consistency and store the result in $S$
3: **if** $S = \emptyset$ **then**
4: Call $\mathbf{MI\_REQ}(CM, |REQ|)$ and store the inconsistent requirements in $IncReqs$
5: Call $\mathbf{askUser}(\mathbf{IncReqs})$ and store the answer in $AdaptedReqs$
6: $CM = KB \cup (REQ \setminus IncReqs \cup AdaptedReqs)$
7: **go to** Step 2
8: **end if**
9: **while** $|S| > n$ **do**
10: Call $\mathbf{GetBestEntrAttr}(A_S)$ and store the result in $a$
11: $A_S = A_S \setminus a$
12: Call $\mathbf{askUser}(\mathbf{a})$ and store the answer in $v_a$
13: $S = \mathcal{R}(S, v_a))$
14: **end while**
15: **return** $S$

and returning the set of recommendations $S$. If no recommendation was found (the empty set is returned), then we have to identify the minimal set of inconsistent requirements. For this purpose, we call algorithm 2 $\mathbf{MI\_REQ}(CM, |REQ|)$ . Algorithm 2 starts with assuming one faulty requirement ($i = 1$) and continues to search, if necessary, up to the number of existing requirements. The constraint solver is this time called restricting the solutions to the specific cardinality $i$ (see Line 2). In Line 3, the function $\mathcal{P}$ is assumed to map the output of the solver to a set of solutions. The termination criteria before reaching $|REQ|$ is given in Line 4, where a non-empty solution obtained from the satisfiability check is returned as result. In case no solution is found, the empty set is returned (Line 8).

**Algorithm 2 MI_REQ**$(CM, |REQ|)$

**Input:** A constraint model $CM$ and the cardinality of the requirements set $|REQ|$
**Output:** Minimal set of inconsistent requirements

1: **for** $i = 1$ **to** $|REQ|$ **do**
2: $M = CM \cup \left\{ \sum_{j=0}^{|REQ|} ab_j = i \right\}$
3: $\Delta_S = \mathcal{P}(\mathbf{CSolver}(M))$
4: **if** $\Delta_S \neq \emptyset$ **then**
5: **return** $\Delta_S$
6: **end if**
7: **end for**
8: **return** $\emptyset$

When being back into the **EntRecom** algorithm, we call the function $\mathbf{askUser}$ in order to adapt the inconsistent requirements ac-

cording to customer preferences. Afterward the constraint model is updated, by mapping the new adapted requirements, and the solver is called once again for checking consistency. In Step 9, the algorithm checks repeatedly if the cardinality of the computed recommendations is greater than the predefined maximum number of recommendations. Within this loop, we first determine the attribute with the best entropy, by calling function $\mathbf{GetBestEntrAttr}$ and store the result in $a$. Note that the entropy of each attribute is computed considering the values from the current set of solutions $S$. Next, we update the remaining set of attributes, then ask the user again about the preferred values of attribute $a$, and store the answer in $v_a$. In Step 13, function $\mathcal{R}$ keeps only the recommendations where attribute $a$ takes values from $v_a$. Algorithm 1 obviously terminates, assuming that **CSolver** terminates.

**Algorithm 3** $\mathbf{GetBestEntrAttr}(A_S)$

**Input:** The set of attributes $A_S$, containing the attributes and their domains accessible using the function $dom$.
**Output:** $a_{res}$ the attribute with the highest entropy

1: $a_{res} = null, ent = -1$
2: **for** $a \in A_s$ **do**
3: $e = \sum_{x \in dom(a)} -P(x) \log P(x)$ compare Equation 1
4: **if** $ent < e$ **then**
5: $ent = e$
6: $a_{res} = a$
7: **end if**
8: **end for**
9: **return** $a_{res}$

Algorithm 3 **GetBestEntrAttr** determines the first attribute having the highest entropy. The algorithm uses the set $A_S$ providing the the domain $d_a$ for each attribute $a \in A_S$. **GetBestEntrAttr** iterates over the set of attributes. In every step it calculates the entropy for the current attribute $a$. If this attribute has a higher entropy than the entropies of the previously selected attributes, this value is stored in $ent$. In addition, the attribute itself is stored in $a_{res}$. After the end of the iteration cycle, the attribute with the highest entropy value stored in $a_{res}$ is given back as result. Obviously, the algorithm terminates providing a finite set of attributes.

With the provide algorithms a chatbot for recommendations can be build that is able to deal with inconsistent requirements as well as missing requirements in a more or less straightforward way making use of previously invented algorithms. We are currently implementing the algorithms into a chatbot environment in order to provide a solid experimental platform for carrying out different case studies.

## 4 Related Work

The use of model-based reasoning and model-based diagnosis in particular in the field of recommender systems is not novel. Papers like [7, 14, 21] compute the minimal sets of faulty requirements, which should be changed in order to find a solution. There the authors compute the diagnosis for inconsistent requirements, relying on the existence of minimal conflict sets. In [7], an algorithm that calculates personalized repairs for inconsistent requirements is presented.The algorithm integrates concepts of MBD with ideas of collaborative problem solving, thus improving the quality of repairs in terms of prediction accuracy. [21] introduces the concept of representative explanations, which follow the idea of generating diversity in alternative diagnoses informally, constraints that occur in conflicts should as well be included in diagnoses presented to the user. Instead of computing all minimal conflicts within the user requirements in advance, [14] proposes to determine preferred conflicts "on demand" and use a general-purpose and fast conflict detection algorithm for this task.

Among the authors who integrate diagnosis and constraint solving more closely, we may mention [5] and later on [26, 27], who proposed a diagnosis algorithm for tree-structured models. Since all general constraint models can be converted into an equivalent tree-structured model using decomposition methods, e.g., hyper tree decomposition [10, 11], the approach is generally applicable. [28] provides more details regarding the coupling of decomposition methods and the diagnosis algorithms for tree-structured models. Further on [23] generalized the algorithms of [5] and [26]. In [18] the authors also propose the use of constraints for diagnosis where conflicts are used to drive the computation. In [9], which is maybe the earliest work that describes the use of constraints for diagnosis, the authors introduce the using constraints for computing conflicts under the correctness assumptions. For this purpose they developed the concept of constraint propagation. Despite of the fact that all of these algorithms use constraints for modeling, they mainly focus on the integration of constraint solving for conflict generation, which is different to our approach. For presenting recommendation tasks as constraint satisfaction problem, we refer to [15].

Human-chatbot communication is a broad field. It includes the technical aspect as well as psychological and human aspects. Papers like [12, 31] show several approaches of implementing chatbots in several domains. [31] shows an artificial intelligence natural language robot (A.L.I.C.E.), as an extension to ELIZA [32], which is based on an experiment by Alan M. Turing in 1950 [30]. This work describes how to create a robot personality using AIML, an artificial intelligence modelling language, to pretend intelligence and self-awareness. In [12] the authors demonstrate the usage of chatbots in the field of tracking food consumption. Sun et al. [29] introduced a conversational recommendation system based on unsupervised learning techniques. The bot was trained by successful order conversations between user and real human agents.

Papers like [8, 13, 16, 33] address the topics user acceptance and experience. In [33] a pre-study shows that users infer the authenticity of a chat agent by two different categories of cues: agent-related cues and conversational-related cues. To get an optimal conversational result the bot should provide a human-like interaction. Questions of conversational UX design raised by [8] and [19] demonstrate also the need to rethink user interaction at all. The topic of recommender systems with conversational interfaces is shown in [17], where an adaptive recommendation strategy was shown based on reinforcement learning methods.

## 5 Conclusions

In this paper, we introduced and discussed a recommendation algorithm based on the concepts of model-based diagnosis and Shannon's

information entropy. The algorithm is intended to be used in a chatbot environment for the tourism domain to handle the user responses via a textual user interface. We presented the challenges to solve common problems in the decision process of a tourist who communicates with such a chatbot. The identified challenges included the case of too many offerings that are presented to the user during the recommendation process and the case of too less offerings, which are caused by inconsistencies between the available knowledge of the chatbot and the given user requirements obtained during a conversation session.

In the proposed approach, we use model-based diagnosis to resolve the inconsistent requirements problem and Shannon's information entropy for solving the issue of too large amounts of offerings by presenting attributes and their values that can be chosen by the user in order to restrict the number of recommendations. Both solutions can be easily integrated within a chatbot environment guiding the chatbot application during the recommendation process.

We are currently implementing the presented algorithms including an integration with an existing chatbot environment dealing with tourism recommender systems. The algorithm is purposed to be used in several other industries and service domains as part of our future work. In the future, we will use this implementation for carrying out experiments and user studies with the objective to show that the approach can be effectively used in practical chatbot settings.

## Acknowledgements

Research presented in this paper was carried out as part of the AS-IT-IC project that is co-financed by the Cooperation Programme Interreg V-A Slovenia-Austria 2014-2020, European Union, European Regional Development Fund.

## REFERENCES

[1] Chatbot Market Size And Share Analysis, Industry Report, 2014 - 2025. https://www.grandviewresearch.com/industry-analysis/chatbot-market. Accessed: 2018-05-07.

[2] Gartner Top Strategic Predictions for 2018 and Beyond. https://www.gartner.com/smarterwithgartner/gartner-top-strategic-predictions-for-2018-and-beyond. Accessed: 2018-05-07.

[3] Johan de Kleer and Brian C. Williams, 'Diagnosing multiple faults', **32**(1), 97–130, (1987).

[4] Rina Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.

[5] Yousri El Fattah and Rina Dechter, 'Diagnosing tree-decomposable circuits', in *Proceedings* $14^{th}$ *International Joint Conf. on Artificial Intelligence*, pp. 1742 – 1748, (1995).

[6] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', **152**, 213–234, (02 2004).

[7] Alexander Felfernig, Gerhard Friedrich, Monika Schubert, Monika Mandl, Markus Mairitsch, and Erich Teppan. Plausible repairs for inconsistent requirements., 01 2009.

[8] Asbjørn Følstad and Petter Bae Brandtzæg, 'Chatbots and the new world of hci', *interactions*, **24**(4), 38–42, (June 2017).

[9] Hector Geffner and Judea Pearl, 'An Improved Constraint-Propagation Algorithm for Diagnosis', in *Proceedings* $10^{th}$ *International Joint Conf. on Artificial Intelligence*, pp. 1105–1111, (1987).

[10] Georg Gottlob, Nicola Leone, and Francesco Scarcello, 'Hypertree Decomposition and Tractable Queries', in *Proc. 18th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS-99)*, pp. 21–32, Philadelphia, PA, (1999).

[11] Georg Gottlob, Nicola Leone, and Francesco Scarcello, 'A comparison of structural CSP decomposition methods', *Artificial Intelligence*, **124**(2), 243–282, (December 2000).

[12] B. Graf, M. Krüger, F. Müller, A. Ruhland, and A. Zech, 'Nombot - simplify food tracking', volume 30-November-2015, pp. 360–363. Association for Computing Machinery, (2015). cited By 3.

[13] Jennifer Hill, W. Randolph Ford, and Ingrid G. Farreras, 'Real conversations with artificial intelligence: A comparison between human-human online conversations and human-chatbot conversations', *Computers in Human Behavior*, **49**, 245 – 250, (2015).

[14] Dietmar Jannach. Finding preferred query relaxations in content-based recommenders, 04 2008.

[15] Dietmar Jannach, Markus Zanker, and Matthias Fuchs, 'Constraint-based recommendation in tourism: A multiperspective case study', *Journal of IT and Tourism*, **11**, 139–155, (2009).

[16] A. Khanna, M. Jain, T. Kumar, D. Singh, B. Pandey, and V. Jha, 'Anatomy and utilities of an artificial intelligence conversational entity', pp. 594–597. Institute of Electrical and Electronics Engineers Inc., (2016). cited By 0.

[17] Tariq Mahmood, Francesco Ricci, and Adriano Venturini, 'Learning adaptive recommendation strategies for online travel planning', *Information and Communication Technologies in Tourism 2009*, 149–160, (2009).

[18] Jakob Mauss and Martin Sachenbacher, 'Conflict-driven diagnosis using relational aggregations', in *Working Papers of the 10th International Workshop on Principles of Diagnosis (DX-99)*, Loch Awe, Scotland, (1999).

[19] R.J. Moore, R. Arar, G.-J. Ren, and M.H. Szymanski, 'Conversational ux design', volume Part F127655, pp. 492–497. Association for Computing Machinery, (2017). cited By 1.

[20] Iulia D. Nica and Franz Wotawa, 'ConDiag – Computing minimal diagnoses using a constraint solver', in *Proc. 23rd International Workshop on Principles of Diagnosis (DX)*, (2012).

[21] Barry O'Sullivan, Alexandre Papadopoulos, Boi Faltings, and Pearl Pu, 'Representative explanations for over-constrained problems', **1**, (07 2007).

[22] Raymond Reiter, 'A Theory of Diagnosis from First Principles', **32**(1), 57–95, (1987).

[23] Martin Sachenbacher and Brian C. Williams, 'Diagnosis as semiring-based constraint optimization', in *European Conference on Artificial Intelligence*, pp. 873–877, (2004).

[24] C. E. Shannon, 'A mathematical theory of communication', *Bell system technical journal*, **27**, (1948).

[25] B. Abu Shawar and E. Atwell, 'Using corpora in machine-learning chatbot systems', in *International Journal of Corpus Linguistics, vol. 10*, (2005).

[26] Markus Stumptner and Franz Wotawa, 'Diagnosing Tree-Structured Systems', in *Proceedings* $15^{th}$ *International Joint Conf. on Artificial Intelligence*, Nagoya, Japan, (1997).

[27] Markus Stumptner and Franz Wotawa, 'Diagnosing tree-structured systems', *Artificial Intelligence*, **127**(1), 1–29, (2001).

[28] Markus Stumptner and Franz Wotawa, 'Coupling CSP decomposition methods and diagnosis algorithms for tree-structured systems', in *Proceedings of the* $18^{th}$ *International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 388–393, Acapulco, Mexico, (2003).

[29] Y. Sun, Y. Zhang, Y. Chen, and R. Jin, 'Conversational recommendation system with unsupervised learning', pp. 397–398. Association for Computing Machinery, Inc, (2016). cited By 0.

[30] Alan M. Turing, *Computing Machinery and Intelligence*, 23–65, Springer Netherlands, Dordrecht, 2009.

[31] R.S. Wallace, *The anatomy of A.L.I.C.E.*, Springer Netherlands, 2009. cited By 53.

[32] J. Weizenbaum, 'Eliza-a computer program for the study of natural language communication between man and machine', *Communications of the ACM*, **9**(1), 36–45, (1966). cited By 1052.

[33] N.V. Wünderlich and S. Paluch, 'A nice and friendly chat with a bot: User perceptions of ai-based service agents'. Association for Information Systems, (2018).

# The Effect of Default Options on Consumer Decisions in the Product Configuration Process

**Yue Wang**[1*] and **Daniel Yiu-Wing Mo**[1]

**Abstract.** Product configurators have been accepted as an important enabling toolkit to bridge customer needs and company offerings. In the configuration process, customers choose from a set of predefined attributes and their options. The combination of choices forms the desired product configuration. It is observed that some online configurators provide default options for each attribute. Although previous studies show that the default option significantly affects customers' choices during the product configuration process, it is not clear how other factors mediate this impact. In this paper, we investigate how product types, number of choices, customers' degree of expertise, the importance of the attributes and the configuring sequence affect consumers' decisions in the configuration process when default options are presented. Based on a series of empirical experiments, we find that customers' degree of expertise, the rating of the attribute importance, and the number of attribute choices have a significant effect on customers' choices for utilitarian products. For hedonic products, the importance of the attributes and the configuring sequence are significant factors.

**Keywords:** status quo effect, configurator, default option, customisation

## 1 INTRODUCTION

Due to the rapid growth of the Internet and e-commerce over the past ten years, online choice configurators have become an important toolkit for customisation by customers. This configure-to-order-based mechanism has been widely used in industry. Successful cases include Dell computers, Adidas, and Nike. By using configuration systems, firms can increase their profit through better sales and higher flexibility. Greater customer involvement in the choice configurator also increases customer satisfaction [1]. Thus, companies can improve their competitive advantage and position by using these toolkits [2].

However, some challenges persist. One of the major challenges is to provide a more user-friendly interface to facilitate choice navigation and decision making in the configuration process. Some effort has been devoted to this research direction. For example, Wang et al. proposed information theory and game theory based method to elicit customer needs adaptively [3] [4]. The configuration sequence is also customised based on the active customer's previous specifications during the configuration process. In this way, the customers' choice navigation process is more efficient and more user friendly. Customers can get what they want quickly and with less burden of cognitive load. Studies have proposed needs-based configuration systems facilitate consumer decision making, particularly for customers without much domain knowledge [5]. The needs-based configurators show a series of product descriptions to customers. Customers then just need to indicate importance or relevance of the descriptions and use semantic words (e.g., 'cheaper' or 'larger') to modify an existing reference product. This can greatly reduce the semantic gap between customer needs and the company's offerings, although the needs in natural language is still not supported.

To help customers make easy decision, default options have been provided in many commercial configurators since mid-1990. Studies also found that the default could potentially help predict customers input when using an interactive online platform [13]. Recently, it has been observed that some online B2C configurators provide default options as well. If a customer makes no choice on the attribute, the default option is selected in the final product, as can be found in the Mini Cooper's configurator in Figure 1.

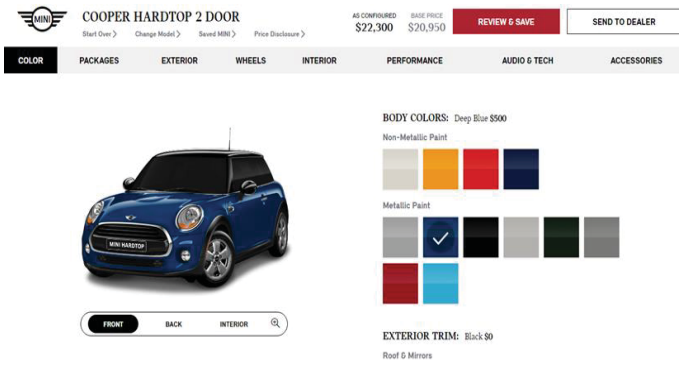


**Figure 1.** Screenshot of the Mini Cooper's online product configurator with default choices (accessed May 2017)

In the study of economics and psychology, it has been acknowledged that the current situation (status quo) is often considered a reference point from the decision makers' point of view. Deviation from the status quo is considered a loss, a phenomenon called 'status quo bias'. According to Mandl and Felfernig [6], status quo bias exists in product configurators, meaning that consumers' decisions are affected by the default options.

Default options have also been studied in the marketing literature. They are considered a type of decision-making heuristic through which cognitive load can be significantly reduced [7][11]. Through empirical experiments, Johnson et al. also noticed that a lack of cognitive attention leads customers to select default

[1] Department of Supply Chain and Information Management, School of Decision Sciences, Hang Seng Management College, Hong Kong, China
*correspondence author, email: yuewang@hsmc.edu.hk

choices. Customers may be paying little or no attention when they choose the default option [7][12]. This type of default is considered an attention-based default.

Brown and Krishna argued that the default options can contain information about the product and thus affect consumer decision making, i.e., they can be considered information-based defaults [8]. For example, they found that low (less expensive) defaults sometimes have more positive effects than high (more expensive) defaults in the case of information-based defaults. In addition, they may create negative effects when customers already know that the default option is the best choice. In this case, customers may be less likely to choose the default choice than the non-default choice.

Compared with expert customers, novice customers more easily accept the default options [9]. Because the complexity of custom decision-making tasks decreases the willingness of customers to participate and reduces the perceived value of the products, novice customers are more affected. This means that when customers are less familiar or have little knowledge of the product, the default options have a greater impact [10].

Although default options have been studied in marketing science research, it is not clear how the default options affect consumers' decisions or which factors are significant in the selection of default choices, particularly in the context of product configuration. Therefore, this paper addresses these questions through empirical experiments. This content is organised as follows. The factors which potentially mediate consumers' decision making under default option setting are introduced in section 2. Section 3 elaborate the design of the empirical experiment. Experimental results and discussion are in section 4. Section 5 concludes the whole paper.

## 2 POTENTIALLY SIGNIFICANT FACTORS

In response to the research question, we conduct empirical experiments to identify the significant factors in customer decisions when default options are presented. The literature suggests that default options affect customers' decisions. However, the process and context of product configuration are different from the product selection process studied in previous research. More factors are involved in the configuration process.

Product type - Products can be classified into two categories: utilitarian products and hedonic products [14]. For utilitarian products, customer choices are based purely on the functional requirements. A certain domain knowledge or expertise is needed to finish the configuring task. For hedonic products, customers' choices are made based their subjective preferences. For example, the corresponding attributes may be colour, shape or design. Customers' preferences for these attributes are subjective. In our research, we ask whether product type mediates customers' selection of default options.

Expertise - Experts have more experience and knowledge of the product, and therefore they may not be affected by the default option because they know what they want to purchase. Unlike experts, novice customers have less knowledge about the product, so they are easily affected by the default option.

Number of choices - it has been acknowledged that the number of choices may also affect consumers' decisions. For example, if an attribute has a large number of choices, the cost of evaluating them may be very high. In this case, customers may use the default options to save effort in the configuring process.

Order of the attributes – Levav showed that the order of the attributes also affects customers' decisions in product customisation [15]. In the present study, the order of the attributes in configurators is considered as a potentially significant factor in customers' choices when they face flexible option configurators.

Concern about the attribute - if a customer cares more about one particular attribute, he or she will be more motivated in the information processing task [16]. Often, consumers do not have enough mental capacity to evaluate all of the attribute levels for all of the attributes offered [17]. Consumers usually start with the most important attribute and proceed based on the order of the attributes' importance [18]. In the context of product configurators, concern about each product attribute is potentially a significant factor in customers' choices.

## 3 EXPERIMENT DESIGN

We develop configurators for a watch and a laptop, which are a hedonic product and a utilitarian product, respectively. Screenshots of the watch and laptop configurators are shown in Figure 2. We only include the components related to aesthetics to the watch configurators. Thus, all of the attributes of the watch can be considered hedonic attributes, meaning that customer choices are based purely on their subjective preferences. No expertise in watches is needed to finish the configuring task. For laptop, we only include the functional components in the configurators. Thus, the laptop's attributes are utilitarian. The choices are determined by customers' functional requirements. A certain amount of background knowledge is needed to finish the configuring task. Because the purpose of this paper is to study which factors affect customer decisions when default choices are presented and customers' satisfaction with the configured product and the configuring process, the comparative study is conducted using a traditional configurator. Thus, the four types of configurators used in this paper are developed as shown in Table 1. For each product, the base configurator is the normal version without default options. This is the configurator used as the control group. For the other versions, each attribute has a default option. To eliminate the effect of option difference on customers' choices, we ***randomly*** assign the default options for each experiment participant. It means that for difference customers, the default options encountered in the configuration tasks are different as well. This configurator is used to investigate consumers' decision behaviour. The default option for each attribute is also randomly selected for each experiment subject. This could offset the influence of choice on consumers' selections.

In the experiment, a participant is randomly assigned to one of the four configurators. After the configuring task, the participant is directed to another configurator with a different product type and configurator type. For example, if the first randomly assigned configurator is configurator III, which is a traditional watch configurator, then the next configurator the participant encounters is configurator II, which has different product type and configurator type. Before each configuring task, the participant completes a pre-experiment survey for each product. The pre-experiment survey is used mainly to determine the relative importance that customers concern about each attribute and their degree of expertise with the utilitarian product. The detailed

questions of the survey are shown in Figure 3. The experiment can be summarised as in Figure 4.

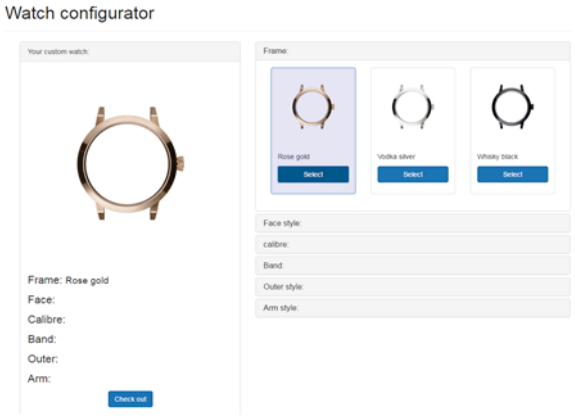


**Figure 2(a).** Screenshot of the watch configurators, with default options

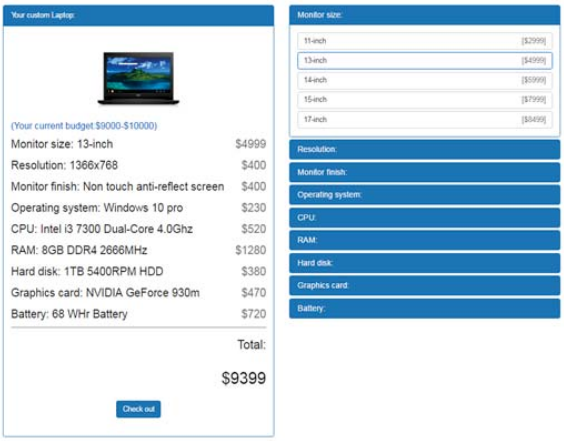


**Figure 2 (b).** Screenshot of the PC configurators, with default options

**Table 1.** Configurators used in the experiment.

| | Base configurator w/o default options | Configurator w/ default options |
|---|---|---|
| **Laptop (utilitarian product)** | I | II |
| **Watch (hedonic product)** | III | IV |

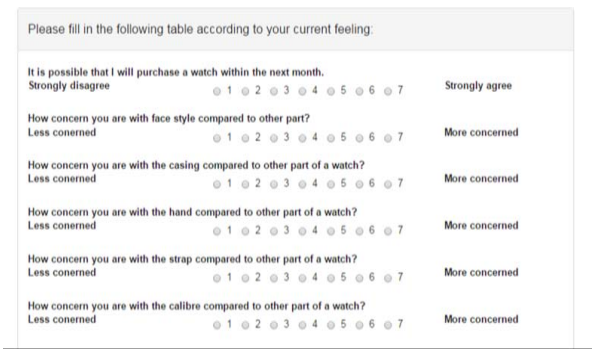


**Figure 3(a).** Screenshot of the pre-experiment survey of watch configurators

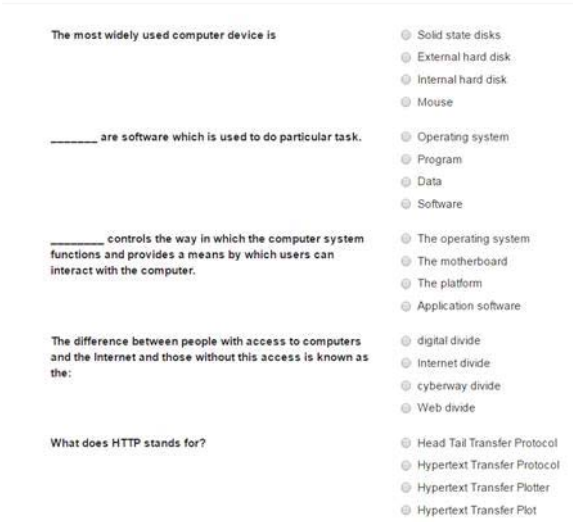


**Figure 3(b).** Screenshot of the pre-experiment survey (partial) of laptop configurators to determine customers' degree of expertise

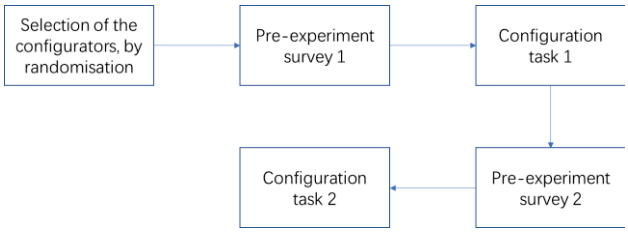


**Figure 4.** Experiment process

# 4 EXPERIMENTAL RESULTS AND DISCUSSION

## 4.1 Basic statistics

One hundred forty participants are recruited from a university in Hong Kong. Each experiment subject receives 30 Hong Kong dollars as compensation for his or her time and effort. We check customers' choice distribution with and without default choices. The purpose is to see whether the default choices lead to a significant difference in consumers' behaviour.

The statistics on the choice distribution are shown in the following table. If the default options have no effect on customers' decisions, the distribution of customers' choices should not be significantly different for the two types of configurators, i.e., with and without default choices. A chi-square test is used to check the difference between the distributions. The p-value of the test result is shown in the last column.

**Table 2.** Consumers' choice distribution for watch attributes

| Attribute | Number of Attribute choices | Attribute choice distribution (with default option, 40 subjects) | Attribute choice distribution (w/o default option, 52 subjects) | P-value of chi-square test |
|---|---|---|---|---|
| Frame | 3 | (15, 14, 11) | (21, 14, 17) | 0.501 |

| | | | | |
|---|---|---|---|---|
| Band | 6 | (3, 9, 3, 11, 4, 10) | (2, 6, 0, 21, 3, 20) | 0.004 |
| Calibre | 2 | (10, 30) | (21, 31) | 0.047 |
| Outer | 8 | (8, 5, 6, 2, 8, 2, 5, 4) | (9, 9, 6, 2, 10, 8, 7, 1) | 0.014 |
| Arm | 2 | (19,21) | (15, 37) | 0.009 |

**Table 3.** Consumers' choice distribution for laptop attributes

| Attribute | Number of Attribute choices | Attribute choice distribution (with default option, 49 subjects) | Attribute choice distribution (w/o default option, 47 subjects) | P-value of chi-square test |
|---|---|---|---|---|
| Monitor | 5 | (19, 19, 8, 2, 1) | (6, 27, 9, 5, 0) | 0.000 |
| Resolution | 3 | (6, 35, 8) | (8, 31, 8) | 0.64 |
| Screen | 2 | (12, 37) | (20, 27) | 0.011 |
| Operating System | 4 | (16,12, 13, 8) | (18, 5, 17, 7) | 0.014 |
| CPU | 6 | (4, 17, 16, 4, 5, 3) | (2, 7, 18, 12, 6, 2) | 0.001 |
| RAM | 9 | (4, 4, 10, 1, 10, 8, 2, 5, 5) | (3, 3, 8, 6, 10, 4, 8, 1, 4) | 0.000 |
| Graphics Card | 5 | (11, 18, 13, 2, 5) | (8, 11, 20, 4, 4) | 0.066 |
| Hard disk | 7 | (7, 11, 4, 8, 8, 5, 6) | (7, 7, 10, 5, 8, 6, 4) | 0.210 |
| Battery | 6 | (5, 8, 12, 8, 5, 11) | (10, 4, 8, 11, 3, 11) | 0.071 |

Based on the tables, we can see that for most attributes, the distributions of customer choices are significantly different, as the corresponding p-value is small. This means that default options affect customers' decisions during the configuring process. We notice that only the watch frame in watch, screen resolution and hard disk in laptop don't have significant difference between the base configurators and the default option-based configurators. After further investigation, we found that the choices for these three attributes either have very strong dominance relationship in terms of customer preferences (screen resolution or hard disk), or very heterogeneous customer preferences (watch frame, the choices can be found in Figure 2). For the former case, customers tend to choose the clearly superior choices regardless of the default options. For the latter case, customers' choices are purely determined by the preferences. Default options can hardly change their intrinsic preferences.

## 4.2 Which factors affect customers' decisions?

Because we want to study the effects of different factors on the selection of default options, it is natural to use a binary variable as an indicator that indicates whether the participant selects the default option in the configuring task for configurators II and IV, as mentioned in the previous section. The independent variables are the number of choices, the order of the attributes, the concern about each attribute and the customers' expertise (only for the laptop, the utilitarian product). The numbers of choices for the two types of products are shown in the second column of Tables 1 and 2. The relative importance that customers accord to each attribute is elicited from the pre-experiment survey. We use the pre-test survey to elicit information about the customers' concern about each attribute. A Likert scale ranging from 1 to 7 is used to allow customers to specify their degree of concern. '1' corresponds to the least degree of concern, and a larger number means a higher degree of concern. A sample question for the watch configurator is 'How concerned are you with the calibre compared to other parts of a watch?' Regarding expertise, we designed a basic knowledge test for laptops containing 10 multiple-choice questions. The number of correctly answered questions is used as the measure of the customer's degree of expertise.

Because the responses are binary variables, logistic regression is used to identify the relationship between independent variables and responses. The result is shown in Tables 3 and 4. For the laptop, the utilitarian product, expertise is an independent variable. For the watch, the hedonic product, the selection of attributes does not depend on customers' expertise; only subjective preferences matter. Thus, expertise is not considered in the regression model of the watch. Model 1 includes all of the independent variables and all of the first-order interactions between independent variables. A stepwise procedure is then conducted to remove the insignificant factors one by one from the model according to the p-value in the regression until only the significant variables remain.

**Table 4.** Relationship between response and different variables - laptop

| Independent Variables | Model 1 (logistic regression) | Model 2 (logistic regression, stepwise result based on model 1) |
|---|---|---|
| Expertise | 0.693*<br>(0.384) | 0.715**<br>(0.321) |
| Concern about attribute | -0.339<br>(0.425) | -0.234***<br>(0.0802) |
| Sequence of configurator | -0.353<br>(0.614) | |
| Number of choices | 0.198<br>(0.424) | 0.402*<br>(0.227) |
| Expertise * Concern | 0.0022<br>(0.0483) | |
| Expertise * Sequence | 0.0082<br>(0.0478) | |
| Expertise * Number of Choices | -0.1053**<br>(0.0515) | -0.1028**<br>(0.0504) |
| Concern * Sequence | 0.0074<br>(0.0494) | |
| Concern * Number of Choices | 0.0093<br>(0.0547) | |
| Sequence * Number of Choices | 0.0613<br>(0.0991) | |

*: p-value<0.1; **: p-value<0.05; ***: p-value<0.01

Remark: the numbers represent the coefficients of the corresponding independent variables in the logistics regression. The numbers in the parentheses are the standard deviation of the corresponding coefficients.

Based on the result shown in Table 4, we find that the degree of expertise is moderately significant in affecting customers'

decisions about default choices. The interaction of degree of expertise and number of choices is significant in affecting customers' decisions to choose the default options. Through a stepwise procedure, we can eliminate the insignificant independent variables one at a time. This leads to model 2, which consists only of the significant independent variables. We find that the degree of expertise, degree of concern about each attribute, and the interaction between degree of expertise and number of choices are significant in affecting customers' decisions. In particular, the coefficient of expertise is positive. This means that if a customer's expertise is greater, he or she is more likely to choose the default options. This finding seems different from previous study in [9]. It should be noted that we use logistic regression to identify the relationship between the independent variables and the choice of default options. In [9], the authors study the relationship between the number of selected default options and the expertise degree. Thus the research questions are different. This can explain the difference of the experiment findings.

The sign of the coefficient of degree of concern is negative, indicating that if a customer is more concerned with an attribute, then he or she is less likely to choose the default options. The coefficient of number of choice is positive, meaning that if an attribute has more choices, customers are more likely to choose the default option. It has been acknowledged that when more choices are presented, the burden of choice is much higher. In this situation, customers may stay with the default option to save time and effort in product configuration.

**Table 5.** Relationship between response and different variables - watch

| Independent Variables | Model 1 (logistic regression) | Model 2 (logistic regression, stepwise result based on model 1) |
|---|---|---|
| Concern about attribute | 0.17<br>(0.337) | -0.218*<br>(0.129) |
| Number of Choices | 0.078<br>(0.145) | |
| Sequence | -0.333<br>(0.446) | -0.334***<br>(0.112) |
| Concern * Number of Choices | -0.0182<br>(0.017) | -0.016***<br>(0.00422) |
| Concern * Sequence | 0.021<br>(0.089) | |
| Number of Choices * Sequence | -0.0315<br>(0.053) | |

*: p-value<0.1; **: p-value<0.05; ***: p-value<0.01

Remark: the numbers represent the coefficients of the corresponding independent variables in the logistics regression. The numbers in the parentheses are the standard deviation of the corresponding coefficients.

For the watch configurator, the attributes are not technical. The selection is based purely on appearance, and no knowledge is required for the configuring task. Therefore, there is no individual variable to quantify the degree of expertise. Based on model 1, we find that none of the individual variables are significant. Through a stepwise procedure, the original regression model can be modified to model 2, in which all of the variables are significant. The degree of concern is moderately significant. Configuring sequence and the interaction of concern with number of choices are significant in affecting customers' decisions to choose the default options. We also notice that all of the signs of the coefficients are negative. Therefore, when customers are more concerned with the attribute, they do not choose the default option. This finding is identical to the case of the laptop. However, in contrast to the laptop configurator, the sequence of the attribute in the configuring process is significant. We think the reason is that for the laptop configurator, the numbers of choices for different attributes are quite similar. However, for the watch configurator, the number of choices ranges from 2 to 24. Thus, the sequence is significant in the customer's decision. In addition, it is observed that customers tend to choose the default options that are presented early. We also find that the interaction between concern and number of choices is also significant in affecting the choices.

## 5 CONCLUSION

Product configurator design has been widely studied in the area of engineering. Very little work investigates the effect of default options on consumer decision making during the configuring process. This paper studies whether default options have a significant effect on people's decisions in the context of product customisation. In the settings of product configurators, a default choice is highlighted for each product attribute. During the experiment, we find that some respondents accept the default choices and others reject them. It is of primary interest to study which kinds of products and what type of attributes are influenced most by the default options. Through a set of empirical experiments, we show that customers' choices are significantly influenced by default options. For utilitarian products, we also note that expertise, concern for the product attribute, number of choices and the interaction between expertise and number of choices significantly mediate the default options' effect on customers' choices. However, for hedonic products, concern about the product attribute, order of configuration and the interaction between concern and number of choices are significant factors. From companies' perspective, customers are more likely to select the default options. This could potentially benefit customisers and improve the operations of the company.

This research still has some limitations. The number of subjects can be larger and the subjects have similar background. Thus, only lab experiment is used to conduct the research. To provide more convincing research outcome, field experiment will be carried out. In addition, the methods on quantifying the expertise degree of the subjects is very sensitive to the discrimination of the questions in the pre-survey test. In our future work, we plan to recruit more participants and further polish the questionnaire to quantify the degree of expertise more accurately. Furthermore, the order of configuration may be a significant factor as well. In the future study, we plan to randomise the configurating order for the research.

## ACKNOWLEDGEMENTS

This research is supported by Hong Kong Research Grants Council (Project No. UGC/FDS14/E02/15, for data collection) and (Project No. UGC/FDS14/E07/17, for data analysis).

## REFERENCES

[1] E. Garbarino and S. M. Johnson, 'The Different Roles of Satisfaction, Trust, and Commitment in Customer Relationships', *Journal of Marketing*, 63(2), 70–87, (1999).

[2] F. S. Fogliatto, G. J. Da Silveira and D. Borenstein, 'The mass customization decade: An updated review of the literature', *International Journal of Production Economics* 138(1), 14-25, (2012).
[3] Y. Wang, and M. M. Tseng, 'Attribute selection for product configurator design based on Gini index', *International Journal of Production Research*, 52(20), 6136-6145, 2014.
[4] Y. Wang, and M. M. Tseng, 'Adaptive Attribute Selection for Configurator Design via Shapley Value', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25 (1), 189–199, (2011).
[5] T. Randall, C. Terwiesch and K. T. Ulrich, 'User Design of Customized Products', *Marketing Science*, 26(2), 268–280, 2007.
[6] M. Mandl, A. Felfernig, J. Tiihonen, and K. Isak, 'Status Quo Bias in Configuration Systems', 24th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2010), Syracuse, New York, 105-114
[7] E. J. Johnson, S. Bellman, and G. L. Lohse, 'Defaults, framing and privacy: Why opting in-opting out', *Marketing Letters*, 13, 5–15, (2002).
[8] C. L. Brown, and A. Krishna, 'The skeptical shopper: A metacognitive account for the effects of default options on choice', *Journal of Consumer Research*, 31, 529–539, (2004).
[9] J. Wang, L. Cheng and W. Han, 'The effect of default option on customer decision behavior in product customization', Proceedings of 10th International Conference on Service Systems and Service Management, Hong Kong, 2013
[10] R. Thaler, and C. R. Sunstein, 'Libertarian Paternalism', *American Economic Review*, 93 (2), 175-179, 2003.
[11] J. Park, and M. R. Banaji, 'Mood and heuristics: The influence of happy and sad states on sensitivity and bias in stereotyping', *Journal of Personality and Social Psychology,* 78, 1005–1023, (2000)..
[12] A. Tversky and D. Kahneman, 'Judgment under Uncertainty: Heuristics and Biases', *Science*. 185(4157),1124-31, 1974.
[13] D. Jannach and L. Kalabis, 'Incremental prediction of configurator input values based on association rules - A case study'. In: Proceedings of the International Workshop on Configuration (ConfWS 2011 at IJCAI 2011). Barcelona, Spain, 2011
[14] R. Batra, O. T. Ahtola, 'Measuring the hedonic and utilitarian sources of consumer attitudes', *Marketing Letters* 2(2), 159-170, (1991).
[15] J. Levav, M. Heitmann, A. Herrmann, and S. S. Iyengar, 'Order in Product Customization Decisions: Evidence from Field Experiments', *Journal of Political Economy*, 118(2), 274-299, 2010
[16] D. J. MacInnes, C. Moorman and B. J. Jaworski, 'Enhancing and measuring consumers' motivation, opportunity and ability to process brand information from ads', *Journal of Marketing*, 55, 32-53, 1991.
[17] R. W. Olshavsky. 'Towards a More Comprehensive Theory of Choice', in NA - Advances in Consumer Research Volume 12, eds. Elizabeth C. Hirschman and Moris B. Holbrook, Provo, UT: Association for Consumer Research, 465-470, (1985).
[18] A. Tversky. 'Elimination by aspects: A theory of choice', *Psychological Review*, 79(4), 281-299, (1972).

# Cost Benefit Analysis in Product Configuration Systems

**Sara Shafiee**[1] and **Alexander Felfernig** and **Lars Hvam** and **Poorang Piroozfar** and **Cipriano Forza**

**Abstract.** Companies' reports indicate a mixture of success and failure in Product Configuration Systems (PCS) projects. Moreover, the attention paid to PCS across different industries is increasing. Therefore, more studies are needed to analyze risks, costs, and benefits of PCS. This paper uses real case projects to demonstrate the cost-benefit analysis of PCSs in real industrial setups. Hence, this article quantifies savings in terms of reduced working hours, and the cost implications with reference to development, implementation, and maintenance. The study fills the gap in previous research by addressing what the influence of other factors on gained cost-benefits from PCSs are likely to be. This study aims to explain why some PCS projects are more cost-effective than the others. While there are a number of factors affecting the cost-benefit analysis in PCS, the focus of this study remains mainly on the number of users and complexity of the project. The comparison in the case studies revealed that both factors have a positive direct correlation with the gained cost-benefits from PCSs.

## 1 INTRODUCTION

Product Configuration Systems (PCS) enable companies to develop product alternatives to facilitate sales and production processes [1]. This is achieved through incorporating information about product features, product structure, production processes, costs and prices [2]. PCSs support decision-making processes in the engineering and sales phases of a product, which can determine the most important decisions regarding product features and cost [3]. PCSs affect the company's ability to increase the accuracy of the cost calculations in the sales phase and consequently increases the products' profitability in sales and engineering process [2].

PCSs can bring substantial benefits to companies such as, shorter lead time for generating quotations, fewer errors, increased ability to meet customers' requirements regarding product functionality, use of fewer resources, optimized product designs, less routine work and improved on-time delivery [2], [4]–[6]. Although advantages of PCSs are evident, there are still some difficulties associated with high cost [2], [7] and considerable chances of failure [8] in their implementation projects.

The aim of this paper is to evaluate the influence of different factors on the gained cost-benefits of PCS such as employees' experiences and organizational culture [9][10]. More specifically, the objective of the paper is to evaluate the influence of the two factors on the cost-benefits gained from different PCS projects: (1) number of users and (2) complexity. This study also sets out to find out why some PCSs are more beneficial than the other PCS projects and how the profitability of the PCS projects in the future can be forecasted. Aiming to investigate these effects, the following propositions were developed:

**Proposition 1.** The higher the number of users in PCSs, the higher Return on Investment (ROI) and cost-benefits**.**

**Proposition 2.** The higher the complexity in PCSs, the higher ROI and cost-benefits**.**

Firstly, we calculate the cost of three different projects during their last four years. Secondly, we calculate the cost-benefits during the last four years. In this research, we focus on the saved man-hours in calculating the ROI on multiple case projects in one case company, while investigating different factors influencing the ROI. Then, the data related to the number of users in the last year and the complexity of PCSs is retrieved. Finally, based on the knowledge in the literature and our research propositions, we demonstrate the results using graphs and discuss the findings.

## 2 LITERATURE STUDY

In this section, the relevant literatures for calculating the PCS cost-benefits and PCS complexity are reviewed which will then be utilized for calculating the ROI and PCS complexity in the cases of this study.

### 2.1 Cost benefit analysis for PCS

The results from the literature review shows that by utilizing PCS reduced man-hours and lead-time for generating the specifications is acknowledged in numerous previous research [5], [11]–[28]. Forza et al. [17] demonstrate a reduction in man-hour from 5-6 days to only 1 day through using PCS. Haug et al. [18] elaborate on how man-hours in the configuration process can be reduced by up to 78.4%. Moreover, Hvam et al.'s [25] study indicates that after utilization of PCS at the case company, the lead time required to generate an offer was reduced by 94–99%. The reduction can be traced to automation of routine tasks and elimination of the iterative loops between domain experts, as PCS makes all product knowledge available [29].

Several researches have quantified the benefits of PCS in terms of reduced man-hours, lead-time and improved the quality of product specifications. However, none of the researchers have investigated the factors which are influencing the cost-benefit analysis and why some of the PCS projects are more cost effective than the others. In this research, we focus on the saved man-hours which is a simple and quantified indicator to calculate the ROI to fill a knowledge gap in the literature.

Discussions concerning the unpredicted costs of PCS projects indicate that the rough estimates involved in cost analysis are considered a challenge that needs more attention from academia [30]. The financial benefits of PCS projects should be clear from

[1] Mechanical Engineering Department, Technical University of Denmark, Denmark, email: sashaf@dtu.dk

the beginning, and cost evaluation is important from the initiation phase. Cost-benefit analysis is used to compare the expected costs and benefits for different scenarios and the results from a variety of actions [31]. ROI, which is commonly used as a cost-benefit ratio, is a performance measure used to evaluate the efficiency of a number of different investments [32], and has been used to determine the profitability of PCS projects [10].

## 2.2 Complexity analysis for PCS

To measure the complexity of PCS, Brown et al. [33] categorize them into three major components; 1) execution complexity, 2) parameter complexity, and 3) memory complexity. Execution complexity covers the complexity involved in performing the configuration actions that make up the configuration procedure while the memory complexity refers to the number of parameters that system manager must remember. In this paper, the parameter complexity is the most important category, as it measures the complexity involved in the knowledge that domain expert provides during the creation of the configuration model [33]. Therefore, we assess the parameter complexity in terms of two major parameters inside the PCS: attributes and constraints (Table 1).

**Table 1.** Complexity assessment in terms of parameters in PCS [34]

| | No. attributes | No. constraints |
|---|---|---|
| **Low complexity** | 500 - 1300 | 200-800 |
| **Medium complexity** | 1300-2000 | 800-1200 |
| **High complexity** | >2000 | >1200 |

# 3 RESEARCH METHOD

The relevant literature was reviewed to clarify the present study's position in relation to existing research. This allowed us not only to ascertain whether this research has the potential to add to the existing knowledge but also to identify which parts of the available knowledge are relevant to this study's scope.

Cost-benefit analysis has been performed in different research areas by calculating the saved man-hours, increased sales, improved quality and reduction in errors and defects. To date, there is no research to investigate the factors influencing cost-benefits in PCSs and to answer why some of the PCS projects are significantly more cost effective.

In the current research, the benefit per quote (in man-hours) and the total cost of the projects is provided by the company. The amount of saved man-hours before and after using the configurator and the gained benefits based on the saved man-hours are calculated. In this study, the total cost of each project is calculated as the project cost, which includes the development, implementation and the yearly running cost (such as licenses and maintenance activities) for the last year.

In this research, we use multiple case studies to evaluate two propositions in one ETO (Engineer To Order) company. The company is a chemical company producing catalysts and process plants and the selected three projects are three catalysts types. The reason for choosing one case company is to provide the in-depth data analysis and observed a trend between the selected factors while all the other factors including organizational culture are fixed. The criteria for choosing the three project (three catalyst products) is the maximum similarities between these three PCS projects to be able to keep other factors constant; the required differences for the selected factors (number of users and complexity); the similar users (engineers); Almost the same rate for the using configurators (number of generated quotes); the same IT team and the involvement of similar tasks during development and maintenance; similar setup of the knowledge; similar software and integrations.

The analysis has been performed during the last 4 years at the case company which allows us to benefit from the strength of using multiple case study method [35], [36]. Furthermore, case studies provide researchers with a deeper understanding of the relations among the variables and phenomena that are not fully examined or understood thus far [37], for instance, the factors with an impact on the cost-benefits from PCS projects. There are multiple data sources such as archived documents and triangulated observations.

# 4 CASE STUDIES

The company selected as the case study produces highly engineered products and technology. The market environment is highly competitive, and thus delivery time and costs are critical. The main motivation for implementing the PCS was to reduce the time required to respond to customer inquiries in order to increase the company's overall competitiveness. Hence, in this study the focus is on lead-time reduction that leads to reduction in resources at the company and directly affects the cost implications.

Three selected projects from three different departments with different number of users and complexities were selected. All three projects are comparable as (1) they all are selected from one case company, (2) they are highly engineered-to-order and complex products, (3) they have been in use during the last 4 years to support sales processes, (4) they have totally different cost-benefits results, and (5) they have are different in terms of complexity and numbers of users. Table 2 demonstrates the data related to three selected sales (commercial) PCS projects. The number of users refers to the sum of the personnel at the company who are using the system (e.g. in Case 1, 50 users constantly use the system). The complexity in this research is relatively studied and different complexities in different projects is compared.

**Table 2.** Number of users and complexity per project

| Case Studies | Number of users per PCS | Complexity of the configurator (sum of attributes and constraints) |
|---|---|---|
| **Case 1** | 50 | Medium/High = 3400 |
| **Case 2** | 13 | Medium = 2100 |
| **Case 3** | 10 | Low = 600 |

Table 3 illustrates all the figures related to the gained benefits based on saved man-hours for each project during the last year.

**Table 3.** Calculation of the total benefits in DKK based on the saved man-hours per year

| Case Studies | Number of quotes per year through configurator | Benefit per quote in hours (saved man-hours) | Total benefit per year (just based on saved man-hours) | Total Costs (development + maintenance + licenses) per year | ROI |
|---|---|---|---|---|---|
| **Case 1** | 240 | 10,3 | 987.840 | 527.000 | 90% |
| **Case 2** | 295 | 1 | 118.000 | 157.000 | 25% |
| **Case 3** | 270 | 0,6 | 65.000 | 110.000 | -40% |

## 5 DISCUSSIONS

The case study results demonstrate how the number of the users and complexity of the configurators' projects have an impact on saved man-hours and cost-benefits in PCS projects.

Analyzing the correlation of the number of users to cost-benefits, clarify the fact that if the department is larger and the potential number of users are higher for one specific PCS, then the expected benefit regarding saved man-hours from that configurator is higher (Figure 1). The number of quotations generated for each of the cases the year before are almost the same (Table 3) but Case 1 saves more man-hours which could be because the time and number of the users for quotation process is higher compared to the other cases.

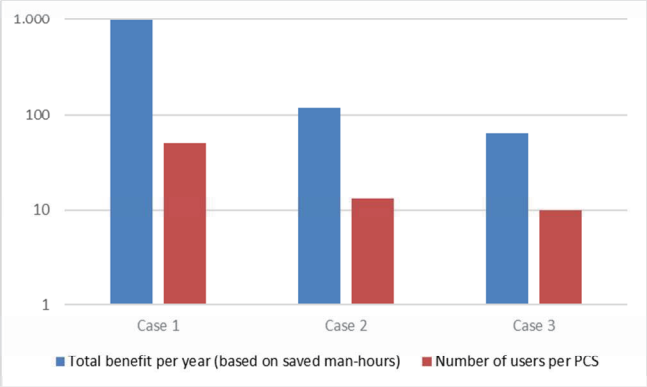


**Figure 1**. The total cost benefits related to the number of users per PCS per year

Analyzing the complexity related to the cost-benefit calculation illustrates a trend in the benefits gained from PCS and their relative complexity ratio. Figure 2 demonstrates a trend between the complexity of the PCS project and cost-benefits implications.

The complexity is calculated based on the attributes and constraints in each project and shows the size of the product as well. The results demonstrate that if the company develops a PCS for more complex product, the project cost will be higher (Table 3), and the benefits will be higher conclusively.

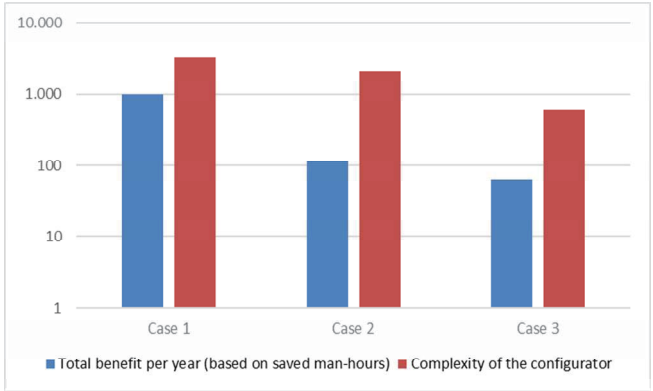


**Figure 2.** The total cost benefits related to the PCS complexity PCS per year

Figure 3 demonstrates the total cost-benefits, number of users and complexity of each case project in one year. As discussed before, there is a direct positive correlation between cost-benefit analysis and both the number of users and the complexity of the project.

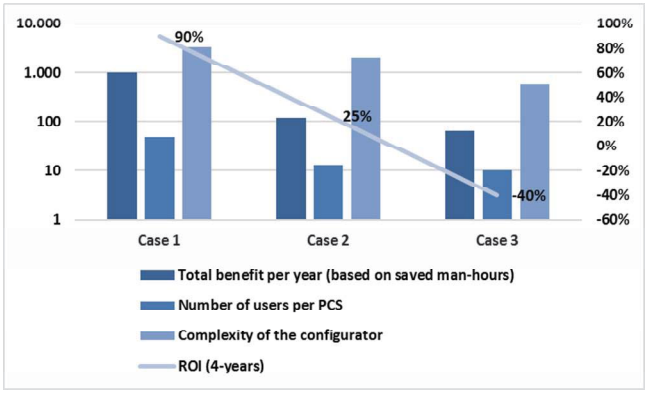


**Figure 3**. The total cost benefits, number of users, complexity per PCS per year

## 6 CONCLUSION

The aim of this study was to measure the influence of the number of users and the complexity of the PCS project on gained benefits based on the same man-hours. The empirical data is gathered from an ETO company based on the previous 4-year results and these results confirmed the propositions. In detail, the gained benefit, number of users, and the PCS complexity per year were measured. The number of users' data was available from the case company and the complexity was calculated based on the number of attributes and the number of constraints in PCS. The PCS complexity illustrate the relative complexity in the product. In order to be able to make the sales configurator for each of these products, a specific number of input, outputs, and finally attributes, constraints, and rules are required in PCS.

The analysis led to the conclusion that there is a positive correlation between the number of users in one PCS and the level of direct savings. The higher number of the employees indicates that PCS can save more man-hours in that specific department. The more complex the PCS project, the more time is needed for developing the project which has been calculated as ROI. However, it seems complex projects save more man-hours. Complex PCS seem to compensate the development efforts and maintenance hours since in such cases, more stakeholders' time is saved to deliver more complicated quotations.

This research is in the first step in exploring the impact of other factors on the saved man-hours in PCS project. There are lists of factors which can influence the PCS projects cost-benefit analysis which can be explored in the future. These factors may be listed as employees' experiences and users' expertise, level of details included in the configurator, and organizational culture. This study considers two specific factors as outstanding ones based on the experience and verified two propositions. In this study, we provided one case company and three projects with in-depth data and we observed a trend between the selected factors. Therefore, it requires further research and additional cases to analyze different factors which may influence the gained benefits from PCS projects. Further research is required to cover both the variety of companies except the ETOs as well as a wide range of case studies.

## REFERENCES

[1] A. Felfernig, S. Reiterer, F. Reinfrank, G. Ninaus, and M. Jeran,

"Conflict Detection and Diagnosis in Configuration," in *Knowledge-Based Configuration: From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufman, 2014, pp. 73–87.

[2] C. Forza and F. Salvador, *Product information management for mass customization: connecting customer, front-office and back-office for fast and efficient customization*. New York: Palgrave Macmillan, 2007.

[3] L. Hvam, N. H. Mortensen, and J. Riis, *Product customization*. Berlin Heidelberg: Springer, 2008.

[4] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-Based Configuration From Research to Business Cases*. Newnes: Morgan Kaufman, 2014.

[5] C. Forza and F. Salvador, "Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems," *International Journal of Production Economics*, vol. 76, no. 1, pp. 87–98, Mar. 2002.

[6] S. Shafiee, "Conceptual Modelling for Product Configuration Systems," Technical University of Denmark, 2017.

[7] A. Haug, L. Hvam, and N. H. Mortensen, "Definition and evaluation of product configurator development strategies," *Computers in Industry*, vol. 63, no. 5, pp. 471–481, Jun. 2012.

[8] S. Shafiee, K. Kristjansdottir, and L. Hvam, "Business cases for product configuration systems," in *7th international conference on mass customization and personalization in Central Europe*, 2016.

[9] L. L. Zhang, "Product configuration: a review of the state-of-the-art and future research," *International Journal of Production Research*, vol. 52, no. 21, pp. 6381–6398, Aug. 2014.

[10] K. Kristjansdottir, S. Shafiee, L. Hvam, M. Bonev, and A. Myrodia, "Return on investment from the use of product configuration systems – A case study," *Computers in Industry*, vol. 100, no. July 2017, pp. 57–69, 2018.

[11] M. Ariano and A. Dagnino, "An intelligent order entry and dynamic bill of materials system for manufacturing customized furniture," *Computers & Electrical Engineering*, vol. 22, no. 1, pp. 45–60, Jan. 1996.

[12] M. Aldanondo, S. Rougé, and M. Véron, "Expert configurator for concurrent engineering: Cameleon software and model," *Journal of Intelligent Manufacturing*, vol. 11, no. 2, pp. 127–134, 2000.

[13] L. Ardissono *et al.*, "A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems," *AI Magazine*, vol. 24, no. 3, p. 93, 2003.

[14] V. E. Barker, D. E. O'Connor, J. Bachant, and E. Soloway, "Expert systems for configuration at Digital: XCON and beyond," *Communications of the ACM*, vol. 32, no. 3, pp. 298–318, Mar. 1989.

[15] M. Gronalt, M. Posset, and T. Benna, "Standardized Configuration in the Domain of Hinterland Container Terminals," *Series on Business Informatics and Application Systems Innovative Processes and Products for Mass Customization*, vol. 3, pp. 105–120, 2007.

[16] C. Forza and F. Salvador, "Product configuration and inter-firm coordination: An innovative solution from a small manufacturing enterprise," *Computers in Industry*, vol. 49, no. 1, pp. 37–46, Sep. 2002.

[17] C. Forza, A. Trentin, and F. Salvador, "Supporting product configuration and form postponement by grouping components into kits: the case of MarelliMotori," *International journal of mass customisation*, vol. 1, no. 4, pp. 427–444, 2006.

[18] A. Haug, L. Hvam, and N. H. Mortensen, "The impact of product configurators on lead times in engineering-oriented companies," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 25, no. 2, pp. 197–206, Apr. 2011.

[19] M. Heiskala, J. Tiihonen, and K. Paloheimo, "Mass customization of services: benefits and challenges of configurable services," in *Frontiers of e-Business Research (FeBR 2005)*, 2005, pp. 206–221.

[20] M. Heiskala, K. Paloheimo, and J. Tiihonen, "Mass customization with configurable products and configurators: a review of benefits and challenges," in *Mass customization information systems in business*, no. C, Finland: IGI Global, 2007, pp. 75–106.

[21] J. Heatley, R. Agarwal, and M. Tanniru, "An evaluation of an innovative information technology - the case of carrier expert," *Journal of Strategic Information Systems*, vol. 4, no. 3, pp. 255–277, 1995.

[22] L. Hvam, M. Malis, B. Hansen, and J. Riis, "Reengineering of the quotation process: application of knowledge based systems," *Business Process Management Journal*, vol. 10, no. 2, pp. 200–213, 2004.

[23] L. Hvam, "Mass customisation in the electronics industry : based on modular products and product configuration," *International Journal of Mass Customisation*, vol. 1, no. 4, pp. 410–426, 2006.

[24] L. Hvam, "Mass customisation of process plants," *International Journal of Mass Customisation*, vol. 1, no. 4, pp. 445–462, 2006.

[25] L. Hvam, A. Haug, N. H. Mortensen, and C. Thuesen, "Observed benefits from product configuration systems," *International Journal of Industrial Engineering: Theory, Applications and Practice*, vol. 20, no. 5–6. 2013.

[26] T. D. Petersen, "Product Configuration in ETO Companies," in *Mass customization information systems in business*, 2007, pp. 59–76.

[27] J. J. Sviokla, "An Examination of the Impact of Expert Systems on the Firm: The Case of XCON," *MIS Quarterly*, vol. 14, no. 2, p. 127, 1990.

[28] M. Yoshioka, M. Oosaki, and T. Tomiyama, "An Application of Quality Function Deployment to Functional Modeling in a Knowledge Intensive Design Environment," in *Knowledge Intensive CAD Vol 1*, no. January, Springer US, 1996, pp. 300–314.

[29] S. Shafiee, K. Kristjansdottir, L. Hvam, and C. Forza, "How to scope configuration projects and manage the knowledge they require," *Journal of Knowledge Management*, vol. 22, no. 5, pp. 982–1014, 2018.

[30] S. Shafiee, L. Hvam, and M. Bonev, "Scoping a product configuration project for engineer-to-order companies," *International Journal of Industrial Engineering and Management*, vol. 5, no. 4, pp. 207–220, 2014.

[31] A. C. Haddix, S. M. Teutsch, and P. S. Corso, *Prevention effectiveness: a guide to decision analysis and economic evaluation*. Oxford University Press, 2003.

[32] P. A. Lemoine, H. C. Woodard, and M. D. Richardson, "Return on Investment," in *Handbook of Improving Performance in the Workplace, Volume Two: Selecting and Implementing Performance Interventions*, 2010, pp. 302–308.

[33] A. B. Brown, A. Keller, and J. L. Hellerstein, "A Model of Configuration Complexity and its Application to a Change Management System Aaron," *IEEE Transactions on Network and Service Management*, vol. 4, no. 1, pp. 13–27, Jun. 2007.

[34] S. Shafiee, L. Hvam, A. Haug, M. Dam, and K. Kristjansdottir, "The documentation of product configuration systems: A framework and an IT solution," *Advanced Engineering Informatics*, vol. 32, pp. 163–175, 2017.

[35] A. H. Van de Ven, "Nothing is quite so practical as a good theory," *Academy of Management Review*, vol. 14, no. 4, pp. 486–489, 1989.

[36] D. M. McCutcheon and J. R. Meredith, "Conducting case study research in operations management," *Journal of Operations Management*, vol. 11, no. 3, pp. 239–256, Sep. 1993.

[37] J. Meredith, "Building operations management theory through case and field research," *Journal of Operations Management*, vol. 16, no. 4, pp. 441–454, 1998.

# Do You Read Me? On the Limits of Manufacturing Part Numbers for Communicating Product Variety

**Aleksander Lubarski**[1,2] and **Frank Dylla**[2,3] and **Holger Schultheis**[3] and **Thorsten Krebs**[2]

**Abstract.** Manufacturing part numbers (MPNs) are used to communicate product variety both for internal purposes and for external representation to customers. To simplify this communication, MPNs have been originally developed as human-readable abbreviations of those characteristics that uniquely identify a product variant out of a modular system, thus acting both as an identifier and a description. However, the increasing complexity of customer requests forces component manufacturers to expand their product portfolios, thus pushing the descriptive character of a classic MPN to its limits. Ongoing digitalization drives product identification towards fully-digital possibilities. Nevertheless, in reality component manufacturers still rely on MPNs. Against this background, the purpose of this paper is to analyze the cognitive and systemic characteristics of the MPN with regard to the underlying product structure (PS). As a result, we derive evaluation criteria for the quality of mappings from PS to MPN and apply them to typical business use cases. In doing so, we provide the first systematic overview and discussion of factors that determine MPN's utility and usability as well as give practical guidelines for component manufacturers regarding the selection of appropriate MPN types.

## 1 INTRODUCTION

With the growing trend of customization and the overall direction towards a lot size of one, manufacturers are facing new requirements of representing their product portfolio in a clear and structured way, as well as understanding their customers' demands. This is a concern particularly in the business-to-business (B2B) markets, where complex machines (and even machine parts) consist of numerous modular elements, thus offering almost unlimited configuration possibilities to the customers. For example, compared to a BMW series 7 who offer as many as $10^{17}$ possible variants [1], Lenze AG claims to have up to $10^{30}$ configuration possibilities just for their gear motor, each of which resulting into a separate identifier. However, if not structured properly, an unnecessarily high product variety can become counter-productive, since customers can get confused about the differentiation among product variants [2]. In this way, while a modular product structure promises a compromise between customer-driven customization and manufacturer-motivated standardization [3], the structuring and the communication of "what's possible and at what price" is a very important, but also a challenging task.

In this context, alphanumeric strings have been used to communicate product variety both for internal purposes and for external representation to customers [4]. For referring to these strings various denominators are common practice, e.g., type code, order code or ID, part ID or number. For reasons of simplicity, we will use the term *manufacturing part number* (MPN, see Section 2.3) throughout this paper. MPNs are codifications of product

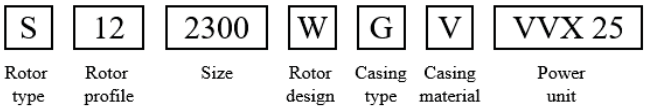


**Figure 1.** Exemplary MPN structure for a rotor system (Lautner GmbH)

characteristics or functions (e.g., information on its structure, material composition, production process) and act as a distinct identifier of a certain product combination (Figure 1). Typical use cases for MPNs range from initial product search to reordering or maintenance of a particular machine. Within this paper, we focus on the appropriateness of such numbers for the purpose of communicating product variety, i.e., visible features. In practice, oftentimes the corresponding stakeholder is provided with relevant supplementary documents that explain the structure and the logic of the MPN and accelerate the process of "deciphering" its information (e.g., tables, graphs, dictionaries). However, although originally developed to be human-readable in order to simplify the customer-supplier communication, we ask the following questions, since many things have changed:

- *Is it still necessary and monetarily reasonable to use MPNs that are human-readable[4]?*
- *Where are the limits of different MPN types and what should the selection of the appropriate MPN depend on?*

In particular, there exist various reasons to switch to a completely digital MPN. First, the descriptive character within any human interaction works only as long as a certain length and complexity of the MPN is not exceeded [5]. For products with a rich variety the number of product characteristics and, consequently, configuration possibilities, is very high. In this way, at some point, the MPN may become too complex and confusing for involved stakeholders. In case a customer is not able to fully understand how a specific MPN relates to the corresponding product may result in delays or incorrect orders, potentially damaging the customer-supplier relationship.

[1] Industrial Services Group, University of Bremen, lubarski@uni-bremen.de
[2] encoway GmbH, {aleksander.lubarski, frank.dylla, krebs}@encoway.de
[3] Bremen Spatial Cognition Center, University of Bremen, {dylla, schulth}@informatik.uni-bremen.de

[4] In this publication we define "human-readable" as the code feature to be read *and* comprehended by the human, while "machine-readable" or "digital" can be processed *solely* by a digital medium

Second, since B2B customers have to deal with dozens of such structures a day from different suppliers, they are no longer willing to invest their time in understanding the internal product structure of each of their partners [6]. This leads to the overall decline of the technical know-how in the market, thus making human-readable MPNs containing these technical details rather obsolete. Third, company mergers and acquisitions may result in inconsistencies and redundancies in the data management[5] [7]. On the one hand, both companies could have similar nomenclatures (or even same serial numbers for different products), thus causing confusion for the future product handling. On the other hand, the customers of the company would get an additional structure to deal with, thus increasing the complexity of variant management and communication.

Nevertheless, despite all advantages mentioned above, it would be narrow-minded to claim that a digital (i.e., only machine-readable) MPN is a universal solution for the efficient communication of the product variety for every manufacturer. The most obvious argument against the abolishment of human-readable MPNs is the complete dependency on a digital device (e.g., QR code reader, barcode scanner) to read and understand the MPN. Examples of situations when this could be problematic include product manufacturing, where a person using an MPN as a guideline for assembly would have to interrupt the production process, or general network coverage problems leading to the inability to use a central referral database. As for other non-functional requirements, the transition to the digital MPN would result in additional hardware-related costs as well as general efforts for restructuring. Moreover, such an initiative could also meet the opposition of the users themselves, who are generally reluctant to any structural changes, since they are used to the way the things are currently handled [8]. Finally, since a digital MPN has all the freedom regarding how much information it can hold and thus communicate, product manufacturers might get tempted to put too much information (if not all of it) in one single digital ID, e.g., a QR-code, thus overwhelming their customers, who would spend additional time for finding the relevant information [9].

Overall, while digital identifiers such as RFID integrated into ERP systems have been proposed more than a decade ago, the reality shows that product and part manufacturers are still far away from the best practice, which calls to find out why. Motivated by the heterogeneity of the involved stakeholders and an overall specificity of product configuration in the B2B markets, we believe that different MPN structures are needed depending on respective application scenarios. Therefore, by applying methods from information theory and insights from cognitive science we analyze relevant characteristics of MPN as well as define evaluation criteria for how well the MPN can map the underlying product structure (PS). By showing that there exist five typical business use cases and thus no universal MPN we contribute to the theoretical discussion on knowledge representation and pave the ground for further research. In addition, as current pragmatic solutions of the practitioners are rather narrow-minded without considering their efficiency, we give practical guidelines for the selection of an appropriate MPN.

The remainder of this paper is organized as follows. After giving a short overview of the theoretical background in Section 2, the aspects of MPN and PS, as well as evaluation criteria of the MPN-PS mapping are explained in Section 3. The evaluation criteria are then applied to the typical business use cases in Section 4. The paper concludes with a summary of the results, possible limitations as well as future research opportunities.

[5] Project experience of the authors shows that these kinds of problems may arise even between units of the same company.

# 2 THEORETICAL BACKGROUND

## 2.1 Code Function and Code Word

In general, a code is an agreement on sets of meaningful symbols for the purpose of information exchange between a sender and single or multiple recipients (e.g., [10], [11]). For this purpose, the sender encodes the information, which needs to be decoded by the recipient with the same coding schema. From a mathematical perspective, a code is an injective mapping from a domain element ($x \in D$) to an element of an image set ($y \in I$). Furthermore, the resulting image must not be empty:

$$f: D \rightarrow I^{+}$$

In the context of codes $D$ and $I$ are considered finite alphabets, i.e. arbitrary sets of symbols with a limited number of elements. In general, codes can be applied for abstracting or abbreviating information and its dependencies. In that sense, not only the Morse code or the internet acronym LOL ("laughing out loud") is a code, but also traffic lights.

From a communication perspective, a code is a translation from the sender's original information or message to some communication means (encoding or encryption), e.g., digital impulses, sound or flags. If the code function is known to the recipient, the original information becomes easily decodable (decoding). Otherwise, the recipient is not able to restore the information at all (no decryption possible). Each code word (a sequence of symbols), which is derivable by the code function, is called a valid code word. Following these definitions, the transfer from a product structure (PS), i.e., simplified a set of product characteristics with certain values, to some part number, e.g. an MPN. The code function is the schema which characteristics are considered and how the corresponding values are represented in the resulting code word, i.e., a product-specific MPN. In the remainder of the paper, in general, we will not distinguish between the code function and the resulting code word and use the term 'code' synonymously.

The application of codes is closely related to *efficiency*, which is achieved by reducing complexity regarding the original information, i.e. by abstraction, abbreviation, or compression. As it takes the effort to design a code, a code gets more efficient the more often it is used. Additionally, some codes use a modular architecture (i.e. decomposition of a complex system in separate functional units) for reasons of efficiency.

With this theoretical background in mind, we consider the relation between code functions and MPN generation from a psychological perspective in more detail in Section 3.

## 2.2 Variant Management

Due to a high level of heterogeneity and a tight customer involvement both in the product design and manufacturing [12],

B2B manufacturers are constantly searching for new ways of standardization without diminishing their ability to go the last mile for their customers [13]. In this context, the introduction of the modular product design (i.e. building a complex system out of exchangeable modules with a clear function and defined connectivity interfaces) has led to a series of organization and production changes, thus expanding configuration possibilities almost exponentially [14]. This resulted in the introduction of such manufacturing concepts as Mass Customization [15] and an overall higher interdependence of the supply chain partners.

However, while a modular product structure promises a compromise between customer-driven customization and manufacturer-motivated standardization [16], the variant management throughout the whole product lifecycle becomes increasingly challenging. A good overview of the sources of complexity in engineering design and manufacturing is given in [5], where complexity is considered as a multi-faceted measurement that is influenced both by endogenous and exogenous drivers.

ElMaraghy et al. [17] define variety as "a number or collection of different things of a particular class of the same general kind", with a variant being an instance of a class that exhibits (slight) differences from the common type. The overall goal of modular product structures is the minimization of *inner variety* while maximizing *outer variety*. In other words – to offer customers as many individualized products as possible with as few parts in production as possible [16]. Additionally, when talking about variant management it is reasonable to differentiate between its two levels – strategic and operative. While the *strategic level* concentrates on "determining and mastering the variety of the product portfolio in such a way that it is aligned with the competitive and the product strategy", the *operative level* implements and secures the overall variant strategy [18]. This publication focuses on the operative level since MPNs are used for the identification and representation of a certain product instance or a category of product combinations, thus supporting the actual implementation of variant management. At this point, we assume that a modularization process has already been conducted and a modular product structure is already given as an input. More information on these steps can be found in [16].

The challenge of operative variant management resulted in the emergence of a specific market for so-called CPQ-systems [19], which can be integrated into existing enterprise software. CPQ software enables product configuration (C), its respective pricing (P) and creation of a unified quote containing all necessary information of the offer (Q). While these quotation documents contain all the details about the desired machine or its component, MPNs are a more compact information representation used for internal and external communication of the product variants and variant identification. As of today, both the topics of CPQ-systems and MPNs have not been devoted enough academic attention, even though they are used in practice on an everyday basis.

## 2.3 Product Identifiers

Since the topic of variant management and product identification involves different research disciplines such as engineering, marketing, information systems or even psychology, there exist no universal definitions or terminology. The challenge lies in the applied nature of the topic and its historical development – not only do the terms and labels differ between companies (especially across different domains and fields of operation), but sometimes even within the same company, depending on the department and the use case. For example, the term "type code" is often used for the description of product groups, even though this term has already established itself in the context of digital storage media, e.g., DVD and Blue-ray [20]. Other misleading terms for the same purpose include "Product Key" or "Type Designation Key", which are used either when installing software programs or when describing specifications and configurations of computer drives [21]. The fact that there exists no standard (international) specification or central point of reference makes it even more confusing for the communication with partners and customers.

Irrespective of the use case, each product or service needs a certain description for the purpose of identification and explanation, containing compressed information in the form of characteristics. Oxford Dictionary defines *Identifier* (ID) as a sequence of characters used to identify or refer to an element, such as a variable set of data. Such an ID can refer either to a unique class of objects with a certain level of abstraction (i.e. products that are grouped based on a certain set of characteristics while, possibly, ignoring further details), or be used for an identification of a specific physical object. A typical example of the object identification is a serial code or *Serial Number* (SN), which is usually comprised of numerals even though other typographical symbols are also possible. In most cases, the SN is built with no particular logic or predefined structure, as it is assigned incrementally or sequentially to an item in the production [22] and thus cannot be read or interpreted by any of the stakeholders. On the contrary, *Manufacturer's Model Number* (MMN) is a solely marketing-driven succinct and catchy description with the purpose of evoking certain associations and interpretations when being communicated to the customers. However, the inevitable problem with the MMN is its necessary level of abstraction – driven by either customer's cognitive capacity (e.g., the ability to remember a certain string) or system limitations (e.g., input fields of ERP systems are often limited in length), the MMN has to concentrate only on the most important product characteristics (e.g., iPhone SE 64 GB black) thus neglecting other valuable product information [4].

Against this background, a compromise between production-driven machine readability and marketing-driven human-readability can be reached by using the term *Manufacturer's Part Number* (MPN) [23], which is the focus of our publication (Figure 2). While also being an identifier for a unique class of objects just like MMN, the MPN contains much more information and is historically built upon a certain predefined structure, which is often sent directly to the customer.[6] Here, an SN is a particular instantiation of an MPN acting as an identifier within a certain context. Therefore, an SN alone cannot be used as a unique identifier (UID), i.e., guaranteed to be unique among all identifiers used for those objects and for a specific purpose, as companies may have identical serial number systems for different products within their product portfolio. Instead, a UID can be created by combining both the MPN and its particular instantiation SN, e.g., "Model X" and "Serial Number 238912". From a mathematical point of view, an MPN is a code word, which

[6] For example, the manufacturing company SEW has a 15-page document showing the structure of their MPN.

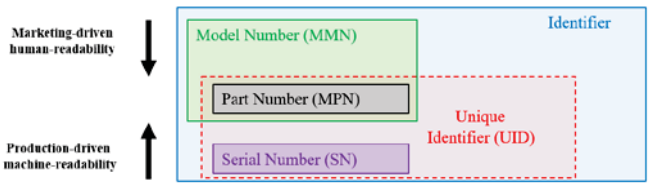


**Figure 2.** Different identifiers and their interrelationship

is derived by a code function from the product structure taking a subset of characteristics into account (cf. Section 2.1).

# 3 MPN-PS MAPPING EVALUATION

## 3.1 Methodology

For the purpose or transparency and traceability of our results, we first give an overview of the methodology used in this paper (Figure 3). It consists of the metaphases *Orientation* and *Application of the MPN*, each containing two consecutive steps along with the applied technique (textual description) and preliminary results (rectangles).

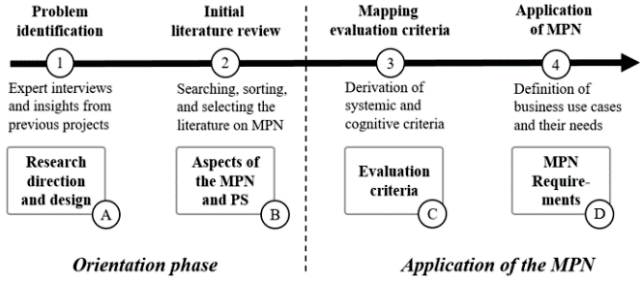


**Figure 3.** Overall paper methodology

Due to the applied nature of our research, first, we contacted the practitioners in order to identify current problems in variant management and product configuration (Step 1). For this, we cooperated with one German consulting company, which specializes in CPQ-Software for complexity reduction and simplification of the quotation process. By conducting semi-structured expert interviews [24] with eight of its previous customers, we were able to gather first insights on the challenges concerning MPN and confirm that different use cases require different types of MPN (Result A).

As a second part of the orientation phase, we then conducted a systematic literature review (Step 2). Sadly, despite its practical importance, the topic of MPN structure and its deployment has not received enough academic attention yet. Therefore, in order to place our research within an ongoing theoretical discussion, we also looked at the adjacent topics of variant management, complexity in manufacturing, and production identifiers. In this way, we were able to identify various MPN aspects, which are relevant for the communication of the product variants (Result B).

The second phase dealt with the application of the MPN in the real-life context. For instance, based on the results from the orientation phase we were able to derive systemic and cognitive evaluation criteria for the MPN-PS mapping in Step 3 (Result C). Since this publication is meant to open a discussion on the topic, we did not provide any specific metrics but remained merely on a conceptual level. Finally, with the help of the conducted expert interviews and based on our own experience from previous projects in Step 4 we defined five business use cases for the MPN deployment, including stakeholder description and requirements for the MPN. In this way, it was possible to show that there exists no universal MPN, but rather that its selection depends on certain industrial, company, and customer characteristics (Result D).

## 3.2 Formal definition

While it may seem that an MPN is just a string of characters, it has numerous systemic and cognitive aspects, which influence its communication and variant management in general. In this context, MPN generation can be seen as a specific mapping $\mathcal{M}$ that is used to represent a given (fixed) product structure in an understandable way (cf. Section 2.1), aiming to be both complete and easy to use (Figure 4). With different product structures and company's specificities, there are many ways to approach such a mapping, ranging from a mere enumeration of all possibilities up to the complex nested configurable structure, meaning that the quality of the mapping can be evaluated for a specific context. We define such an evaluation function $\mathcal{E}$ of mapping $\mathcal{M}$ with respect to a specific need or application of the generated MPN, defined by a set of evaluation criteria $\boldsymbol{c} = [c_1, \dots, c_m]$ and corresponding outcomes $\boldsymbol{x} = [x_1, \dots, x_m]$:

$$\mathcal{E}(\mathcal{M}, \boldsymbol{c}) = \boldsymbol{x} \; with \; x_i \in [0..1]$$

Based on the insights from the expert interviews (Figure 3, Step 1) and literature review (Figure 3, Step 2), in this publication, we concentrate on the general mapping $\mathcal{M}$ and the definition of the corresponding evaluation criteria as well as their testing for different use cases. The subsequent optimization problem, i.e. finding an optimal MPN with respect to a given set of criteria, is beyond the scope of the publication and is left for the future research.

## 3.3 Product and MPN structure

When talking about the evaluation of the MPN it is important to differentiate between two different types of structure (Figure 4).

The *Product Structure* is a rigid arrangement of elements that is used to depict product compositions and configuration possibilities. Unlike the MPN structure, the actual product structure (i.e. what elements a complex machine consists of, what possible combinations are there, etc.) states the core of the value proposition and thus cannot be changed. In this regard, the *Number of Product Characteristics* shows which parts of a complex product can be substituted or exchanged if desired. Similarly, such a modular product composition and interface specifications determine the *Configurable Variety*, thus enabling satisfaction of customer heterogeneous requirements. However, as mentioned earlier, a

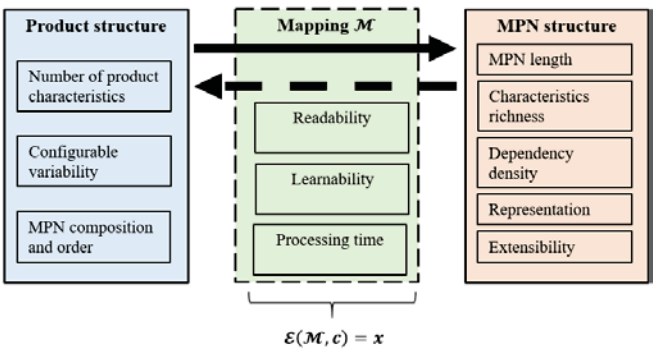


**Figure 4.** Depiction of the product structure with the use of MPN

higher configurable variety does not necessarily mean a better portfolio, since an excessively high variety can even be counter-productive, as customers can get confused about the differentiation among product variants. Finally, the actual *Composition of the MPN* along with the sequential arrangement of the characteristics within is influenced by the product structure. In other words, the stakeholder is able to read and understand the information encoded in the MPN only as long as he/she knows what symbol is responsible for what characteristics.

The *MPN structure*, on the other hand, is flexible and can be adjusted according to the company's preferences. A cleverly picked MPN can enhance human understanding and thus its overall manageability and understandability, whereas a bad MPN may confuse both the company employees as well as their customers. This includes the *Length of the MPN*, which may sometimes be challenging, since many ERP systems (e.g., SAP) have an integrated limit on the string length, thus indirectly forcing product manufacturers to use shorter codes [4]. Another important MPN characteristic is an average *Characteristics Richness* showing how many variants are encoded inside single characteristics. This aspect is especially important if the MPN is human-readable and a person operating it has to remember all the variants of every characteristic by heart. Finally, the amount of configurable feasible solutions is limited by the applicable logical rules (e.g., IF, XOR, AND), which can be summarized by the aspect *Dependency Density*. Although acting as a limitation to the possible solution space, dependency density increases the complexity of the MPN and thus its applicability by a human stakeholder. Looking at the long-term life-cycle of products, another central aspect is the *Extensibility* of the MPN. This is particularly important for products, which are still in the development and it is foreseeable that new characteristics may be added according to customer's preferences or market's development, e.g., by adding functionality by software updates. The level of extensibility can hereby range from practically impossible (e.g., rules defining the MPN structure prohibit any further extension) or cost-inefficient (e.g., requiring database connectivity) to complete freedom (e.g., no limitations within a digital MPN). Finally, the choice of MPN *Representation* type predefines how it will be used between the involved actors. Three possible types of MPN representation can be differentiated, each with its advantage and limitation: (i) human understandable MPN that can be read and assessed without any additional help, meaning that human operator needs to know the MPN structure and characteristic meaning by heart; (ii) human readable MPN that can be read and assessed using a certain (analog or digital) code translator or additive requiring a certain level of know-how of the user; (iii) machine-readable MPN that can contain almost unlimited amount of coded information within product characteristics, without any know-how requirements for the user, but with a high dependency on a digital scanner.

## 3.4 Mapping evaluation criteria

Once an appropriate MPN and a clear mapping of the product structure to the MPN is created, the question arises how well a human can employ it, with or without any auxiliary means, i.e. how complex the code function is w.r.t. human processing (cf. Section 2.1). If we abstract from specific company restrictions or personal preferences, the human user should be able to map the MPN to a corresponding product 1) quickly, 2) with acceptable accuracy, and 3) without extensive learning. A prerequisite for achieving this is that the MPN and the mapping realized by it are conducive to the way in which humans process information. For example, consider the acronym "TDI" to characterize a property of car motors. If confronted with this acronym in the context of cars, people may already "know" that the car being under consideration has a diesel motor with "Turbocharged Direct Injection".[7] At this point, the question arises: how or why do people know this? If observed separately, these three letters "TDI" could also stand for something else, e.g., "Total Dream Interior", thus describing some arbitrary aspect of the car. The only source for this knowledge is the memory of the person having to deal with the acronym. In other words – the working and usability of MPNs for human users relies critically on the memory capacity of the human user.

The human cognitive system is commonly conceived as consisting of several different types of memory stores [25]. For reasons explained in the following, the two types of stores most relevant for our considerations with regard to MPNs are *working memory* (WM) and *long-term memory* (LTM). Information recently having or currently being processed resides in WM. As such, WM is assumed to be limited in capacity as well as in the duration that content will remain accessible. Although exact estimations vary it seems clear that no more than 10 items of information can be maintained simultaneously [26, 27]. If the information in WM is not used, it will be lost within a few seconds [28]. Human LTM, on the other hand, can store vast amounts of information (see [29] for estimates) over very long time spans (just think of old people telling stories from their childhood). In fact, depending on the usage frequency, information from LTM can be retrieved into WM and information residing in WM may be transferred to LTM for more permanent storage. Going back to the "TDI" example, it seems most likely that knowing that "TDI" signifies "Turbocharged Direct Injection" is retrieved from LTM. If it was not the case, it would have had to be maintained in WM since first learning what "TDI" means. This may be the case when someone first tells you what "TDI" means in the domain of car motors, but on later encounters, it seems unlikely that the meaning of the acronym has been constantly maintained in working memory for the whole time. Given that LTM plays a central role in the human use of MPNs, it is instrumental to consider some of the properties of how information is represented and organized in LTM. Two properties seem particularly noteworthy.

First, human LTM is *associative* [30]. Certain concepts/pieces of information are associated with each other such as, for example, the concept of "fire" is associated with the concept of "heat". In particular, these associations are a major means of retrieving information from LTM. If a current item in WM is sufficiently strongly associated with some other piece of information in the LTM, retrieval of this other information is greatly facilitated. If, for instance, "TDI" is sufficiently strongly associated with "Turbocharged Direct Injection" in the context of cars, seeing the acronym will allow retrieving the desired information.

[7] Due to its direct connection to the fuel type Diesel it is also considered as "Turbocharged Diesel Injection".

Second, knowledge organization in human LTM is *hierarchical* [31, 32]. This means that human LTM has a tendency to be organized into categories, subcategories, and concrete instances. If asked to enumerate properties/instances in a certain domain, people will often enumerate aspects (sub)category after (sub)category. Vice versa, if people are asked to memorize a (large) list of properties, the ability to memorize the given information increases considerably, if the material can be (and is) organized hierarchically. Having expounded the crucial characteristics of human memory, we now relate them to the readability, learnability, and processing speed of MPNs.

### *3.4.1 Readability / Learnability*

In the context of this publication, we consider an MPN readable, if a human user can map each MPN to the corresponding product (a) virtually without error and (b) without any supporting material (e.g., tables, software). Thus, the following requirements have to be met:

- The MPN has to be short enough to be held in WM completely or it has to consist of several meaningfully separable parts, each of which is short enough to be held completely in WM.
- The MPN or its meaningful parts have to be sufficiently strongly associated in LTM with the product aspects they represent.

The first requirement seems comparatively easy to achieve. A ten-arity code using letters and ciphers is able to represent about $35^{10}$ different product variants. The second requirement is tightly related to the learnability of the MPN, that is, to the feasibility of acquiring all necessary associations and the speed with which they can be acquired. As a result, the second requirement is much harder to satisfy as soon as we are dealing with large-scale variant spaces. There seems to be an upper limit of only around 5000 arbitrary associations that human LTM can store, while learning a good 3000 arbitrary associations took a whole year (incremental training with three sessions each day) [33]. If the human user is supposed to go beyond such limitations, the code either needs to build on and reuse existing associations or the code has to be organized hierarchically or both. The simplest form of reuse is to employ non-arbitrary mappings. For example, in the case of "TDI" the acronym is much easier to memorize than arbitrary acronyms (e.g., "XYZ"), because the letters in the acronym are the starting letters of the crucial words in the product description and, thus, the letters are already associated with the target words. More elaborate forms of reuse may involve exploiting more domain-specific knowledge of the human users. A hierarchical organization of the code would allow reducing the number of associations that need to be stored at each level of the hierarchy, while still allowing to cover a large variant space with the MPN. For example, the first digit of the MPN may signify whether the product is a car or a motorbike with the remainder of the key then being specific to the type of vehicle (car or motorbike). The upper limit of possible associations that can be stored when drawing on existing knowledge and hierarchical organization is hard to predict and will probably also depend on the individual user. Nevertheless, it is clear that the readability/learnability of the MPN increases:

- The more easily (parts of) the MPN can be maintained in working memory,
- The fewer associations have to be memorized,
- The more previous knowledge can be drawn on, and
- The clearer it can be structured hierarchically.

### *3.4.2 Processing speed*

Processing speed can be assumed to be directly related to the length and the complexity of the MPN: The shorter the MPN and the less complex (i.e., the fewer associations) the faster the human user will be able to map the MPN to the product. Direct mappings can be assumed to be faster than hierarchical mappings, because each level in the hierarchy requires a separate mapping, while a direct mapping – once sufficiently learned – involves only a single mapping process. If a hierarchical MPN is employed, the order in which MPN parts signifying different (sub)categories appear in the MPN will have an influence in processing speed. At least in western culture languages, representing highest to lowest (sub)categories by MPN parts ordered from left to right (the predominant reading direction) will likely lead to faster MPN processing than other orderings. Consequently, the processing speed of MPN increases

- The shorter the MPN,
- The fewer associations are involved,
- The fewer hierarchical levels are involved, and
- The more the ordering of the hierarchy levels in the MPN corresponds to the reading direction.

## 4 USE CASES

Motivated by the diversity of the customer requirements and specificity of product configuration in the B2B markets, we propose five different application scenarios also referred to as “use cases” in the context of business modeling [34]. The use cases are based on the expert interviews and insights from the previous projects (Figure 3). To ensure comparability, we describe each of the use cases using following characteristics: (i) role of the involved actor, (ii) his/her aim and objective, (iii) current know-how level in the field, as well as (iv) what is expected from the desired solution. Consequently, with the help of the previously introduced evaluation criteria, we derive individual MPN requirements for each of the use cases (Table 1). For simplicity reasons, the costs relating to the implementation and maintenance of a specific type of MPN (e.g., establishing the new MPN structure, employee training, structure maintenance) is beyond the scope of this paper but should be considered in future research.

Starting with the customer side, one particular type of users is a *Rare Guest*, who is characterized by a low frequency of MPN use and the need for a specific (one-time) solution. These customers have little to none expert knowledge on the topic, either because of the involvement in various supply chains or due to the overall unwillingness to deal in detail with the product structure of the particular manufacturer. Their expectation towards the communication of product variety is a mere satisfaction of their request with as little effort as possible. In this way, the rare guest would not appreciate the efficiency of the MPN (e.g., through the digitalization), but would be disappointed if standard functionalities are not functioning properly. At the other end of the spectrum, there is a *Power User*. Unlike the rare guest, the power user deals with the

respective MPN on a regular basis. Due to the high level of know-how both regarding the operating industry and the MPN structure, the power user is interested in a fast interaction and comparability of different variants within this particular manufacturer, in order to attain the optimal solution for a specific situation. Typically, power users are established via long-term cooperation between companies and would be interested in a high learnability of the MPN. Moreover, they personally would even get averse to switch to a purely machine-readable MPN, since this would make their own know-how obsolete.

The last external stakeholder of the MPN we consider is the *Purchasing* department, who already has a particular product configuration in mind. Similar to the rare guest, the purchasing department has little interest in the product itself (or its configuration), but mostly focuses on getting the best price and most suitable delivery time. In addition, due to the overall desire for comparability across different manufacturers, the purchasing department seeks understandability and simplicity of the MPN and, if possible, the establishment of an industry-wide standard.

Apart from the customer-driven requirements towards the MPN, manufacturer's internal departments have their own, often contradictory expectations. For example, the *Sales* department uses MPN for selling, searching, and configuring product variants in a simple and quick manner. Similar to the purchasing department of the customer, the internal sales department seeks a practical and catchy MPN structure, which can be used for the external representation of a large quantity of (homogeneous) products. With the level of know-how varying according to the position and the seniority of the sales employees, the efficiency of the sales department is dependent either on the learnability of the human-readable MPN or on the increase of the processing speed due to its digitalization.

Finally, the employees working in the internal *Production* department of the company are looking for an MPN, which would both contain information on the internal structure of the product and have a high level of learnability. Their daily usage of the MPN involves the identification of the required parts and their assembling sequence. With a highly specific background knowledge and fast production processes, production employees would appreciate a digital MPN only if it can assure a low error rate and not obstruct or slow them down in their daily activities.

## 5 CONCLUSION AND OUTLOOK

With the increasing customer demands for individualization and an overall growing complexity of B2B products, part manufacturers are facing new problems of communicating their product variety in a transparent and efficient way. In this context, MPNs are used as codifications of those characteristics that uniquely identify a product variant out of a modular system, thus acting both as an identifier and a description. While academia has not yet devoted enough attention to the topic of MPN, we believe that a correct deployment of the MPN is a key step to efficient variant management. Therefore, in this publication, we analyzed the main systemic and cognitive aspects of the MPN and derived evaluation criteria (i.e. readability, learnability, and processing speed) for its mapping on the underlying product structure. We then mapped these evaluation criteria on various use cases of the internal and external stakeholders, thus showing that there exists no ideal MPN, but instead it should be created depending on the respective application scenario.

This publication also highlights important topics for future research endeavors. First, despite a detailed analysis of the existing literature on manufacturing complexity combined with personal project experience of the authors, we believe that there exist additional aspects of the MPN and the product structure. Similarly,

**Table 1.** Various use cases regarding the deployment of MPNs

| | | *External* | | | *Internal* | |
|---|---|---|---|---|---|---|
| *Use case description* | **Role** | **Rare Guest** | **Power User** | **Purchasing** | **Sales** | **Production / Logistics** |
| | **Goal** | Identification, searching, configuration | Identification, searching, configuration | Acquisition, reorder | Sale, searching, configuration | Overview of the internal structure |
| | **Use Case** | Solution of an application problem | Finding an optimal variant for a specific situation | Best price and delivery time | Suitable product, possible large quantity | Fast identification of the required parts and their assembling. Low error rate |
| | **Know-How** | Low background knowledge | High background knowledge | Little interest in the product | Diverse (internal vs. external sales, seniority) | High background knowledge (highly specific) |
| | **Desired Solution** | Description of the problem, not technical detail | Fast interaction, comparability of the variants | Comparability across different manufacturers | Same description of the homogenous products | Fast transfer (e.g., location of a product part) |
| *MPN requirements* | **Read-ability** | No preferences, as long as it functions right | Human-understan-dable (without any auxiliary means) | Type of readability that enables supplier comparability | Catchy type and structure that can be used for marketing | Human-understandable (without any auxiliary means) |
| | **Learn-ability** | Not needed, since no intentions for reuse | High expectations towards the logic of the MPN structure | Not needed, since no intentions for reuse | High expectations towards the logic of the MPN structure | High expectations towards the logic of the MPN structure |
| | **Process-ing speed** | No preferences, as long as it functions right | Crucial to the efficiency of their everyday work | Preference direction machine-readable MPN | Preference direction machine-readable MPN | Crucial to the efficiency of their everyday work |

future work should differentiate between industries or company-specific characteristics.

Altogether, our publication has no claims to completeness but is intended to serve as a starting point of an academic discussion on the MPN and the communication of the product variety in general. Future research should focus more on the differentiation between different layers of the company, in particular, the production and the sales layer, which may even lead to different MPNs. Another research direction is the operationalization of the above-mentioned mapping by introducing a maturity model [35] or some other decision-support framework. This may also include the task on how to find optimal MPN for a specific set of criteria. A good opportunity for creating such a model would be by following an action research, where the transformation of the company's MPN structure (e.g., towards a digital one) is supervised and documented. Finally, the monetary aspects and the area of cost efficiency in general (e.g., calculation of the possible cost savings, if a certain MPN is introduced), which were not considered in this publication, would make a major contribution, especially for the practitioners.

## REFERENCES


[1] S. J. Hu, X. Zhu, H. Wang, and Y. Koren, „Product variety and manufacturing complexity in assembly systems and supply chains“, *CIRP Ann.-Manuf. Technol.*, Bd. 57, Nr. 1, S. 45–48, 2008.

[2] C. Huffman and B. E. Kahn, „Variety for sale: Mass customization or mass confusion?“, *J. Retail.*, Bd. 74, Nr. 4, S. 491–513, 1998.

[3] C. Y. Baldwin and K. B. Clark, *Design Rules: The Power of Modularity Volume 1*. Cambridge, MA, USA: MIT Press, 1999.

[4] T. Blecker, N. Abdelkafi, B. Kaluza, G. Kreutler, „Mass customization vs. complexity: a Gordian knot?", Munich Personal RePEc Archive, 890-903, 2004.

[5] W. ElMaraghy, H. ElMaraghy, T. Tomiyama, and L. Monostori, „Complexity in engineering design and manufacturing“, *CIRP Ann.*, Bd. 61, Nr. 2, S. 793–814, 2012.

[6] C. Forza and F. Salvador, „Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems“, *Int. J. Prod. Econ.*, Bd. 76, Nr. 1, S. 87–98, März 2002.

[7] P. Mishra and T. Chandra, „Mergers, Acquisitions and Firms Performance: Experience of Indian Pharmaceutical Industry“, *Eurasian J. Bus. Econ.*, Bd. 3, Nr. 5, S. 111–126, 2010.

[8] V. Venkatesh and F. D. Davis, „A theoretical extension of the technology acceptance model: Four longitudinal field studies“, *Manag. Sci.*, Bd. 46, Nr. 2, S. 186–204, 2000.

[9] F. Piller, M. Koch, K. Moeslein, and P. Schubert, „Managing high variety: how to overcome the mass confusion phenomenon of customer co-design“, in *Proceedings of the Proc. 3rd Annual Conf. of the European Academy of Management (EURAM 2003), Milan, Italy*, 2003.

[10] D. W. Hoffmann, *Einführung in die Informations-und Codierungstheorie*. Springer, 2014.

[11] W. Heise und P. Quattrocchi, *Informations-und Codierungstheorie: mathematische Grundlagen der Daten-Kompression und-Sicherung in diskreten Kommunikationssystemen*. Springer-Verlag, 2013.

[12] F. Piller and C. Berger, „Customers as co-designers“, *Manuf. Eng.*, Bd. 82, Nr. 4, S. 42–45, Aug. 2003.

[13] A. Bask, M. Lipponen, M. Rajahonka, und M. Tinnilä, „The concept of modularity: diffusion from manufacturing to service production“, *J. Manuf. Technol. Manag.*, Bd. 21, Nr. 3, S. 355–375, 2010.

[14] C. Y. Baldwin and K. B. Clark, *Design rules: The power of modularity*, Bd. 1. MIT press, 2000.

[15] B. J. Pine, *Mass customization: the new frontier in business competition*. Harvard Business Press, 1999.

[16] A. Lubarski und J. Poeppelbuss, „Methods for Service Modularization–A Systematization Framework“, in *Proceedings of the Pacific Asia conference on information systems (PACIS), Chiayi, Taiwan*, 2016.

[17] H. ElMaraghy *u. a.*, „Product variety management“, *CIRP Ann.*, Bd. 62, Nr. 2, S. 629–652, 2013.

[18] B. Avak, „Variant management of modular product families in the market phase“, ETH Zurich, 2006.

[19] Gartner IT Glossary, „Configure, Price, Quote (CPQ) Application Suites“. 16-Apr-2018.

[20] S. Tsukatani, T. Inokuchi, and H. Ito, „Multi-session disc-shaped for recording audio and computer data having disc type code area located in each session for recording common and particular disc type code“, Juli-1998.

[21] D. E. Denning and D. K. Branstad, „A taxonomy for key escrow encryption systems“, *Commun. ACM*, Bd. 39, Nr. 3, S. 34–40, 1996.

[22] R. Elz, „Serial number arithmetic“, 1996.

[23] J. Schickler, „Parts search system“, Aug-2004.

[24] R. K. Yin, *Case study research and applications: Design and methods*. Sage publications, 2017.

[25] B. J. Baars and N. M. Gage, *Fundamentals of cognitive neuroscience: a beginner's guide*. Academic Press, 2013.

[26] N. Cowan, „The magical number 4 in short-term memory: A reconsideration of mental storage capacity“, *Behav. Brain Sci.*, Bd. 24, Nr. 1, S. 87–114, Feb. 2001.

[27] G. A. Miller, „The magical number seven, plus or minus two: Some limits on our capacity for processing information.“, *Psychol. Rev.*, Bd. 63, Nr. 2, S. 81, 1956.

[28] A. Baddeley, „Working memory: theories, models, and controversies“, *Annu. Rev. Psychol.*, Bd. 63, S. 1–29, 2012.

[29] Y. Dudai, „How big is human memory, or on being just useful enough.“, *Learn. Mem.*, Bd. 3, Nr. 5, S. 341–365, 1997.

[30] J. R. Anderson, *Cognitive psychology and its implications*. Macmillan, 2005.

[31] T. P. McNamara, „Mental representations of spatial relations“, *Cognit. Psychol.*, Bd. 18, Nr. 1, S. 87–121, 1986.

[32] A. M. Collins and M. R. Quillian, „Retrieval time from semantic memory“, *J. Verbal Learn. Verbal Behav.*, Bd. 8, Nr. 2, S. 240–247, 1969.

[33] J. L. Voss, „Long-term associative memory capacity in man“, *Psychon. Bull. Rev.*, Bd. 16, Nr. 6, S. 1076–1081, 2009.

[34] H.-E. Eriksson and M. Penker, „Business modeling with UML“, *N. Y.*, S. 1–12, 2000.

[35] M. C. Paulk, *The capability maturity model: Guidelines for improving the software process*. Addison-Wesley Professional, 1995.

# Behavior-Driven Development in Product Configuration Systems

**Sara Shafiee**[1] and **Lars Hvam** and **Anders Haug** and **Yves Wautelet**

**Abstract.** Product Configuration Systems (PCS) are increasingly used by companies to automate the performance of the sales and engineering processes. Since the benefits from such projects have huge variations, it is crucial to make the right decisions when scoping and developing PCSs. The development of PCS is influenced by both business interests and technical insights. Developers of PCS face various challenges while working in team, including different stakeholders such as business owners, developers, project managers, and product experts. The more diverse the team is, the more significant are the challenges. This paper suggests that Behavior-driven Development (BDD) may provide configuration teams with a specific structure to express scenarios (and thus constraints) on PCS in natural language. BDD may yield benefits such as a better expression of PCS constraints, more efficient communication of requirements and incorporation of the expressed rules in a software transformation process. In other words, applying BDD may eliminate unnecessary tasks when gathering knowledge, developing, and testing PCS projects. In this paper, we present a novel approach from an ongoing project on how to relate BDD to the development process of PCS while using Scrum-based methods.

## 1 INTRODUCTION

Product Configuration Systems (PCS) are expert systems that can support and facilitate the sales and engineering processes [1] by incorporating information about product features, product structure, production processes, costs and prices [2]. Thereby, configurators can support decision-making processes in the sales and engineering phases of a product [3]. Configurators enable companies to develop product alternatives to facilitate sales and production processes [4]. However, the companies that have managed to implement and utilize configurators also face various challenges [1], [5].

Even though advantages of PCS are evident, there are still some difficulties associated with high costs [2], [6] and considerable chances of failure [2], [7] in their implementation projects. This is because companies must overcome various challenges to implement and utilize configurators. For example, the development of a PCS often requires highly complex technical or commercial knowledge, which domain experts often have a hard time communicating to configuration experts [8]. Furthermore, the knowledge base that contains this knowledge has to be adapted continuously because of the changing components and configuration constraints [9], [10]. The difficulty of acquiring and modeling the required technical or commercial knowledge depends on whether it is available in a clear and formal form [11] which in turns may be contingent to company size [12], product complexity [13], degree of customization [14], or other factors such as knowledge management and scoping process [8]. The challenges to manage technical and commercial knowledge when implementing and maintaining a PCS may highly influence PCS costs and development time, as well as its lasting effectiveness.

The complexity in the development of PCSs comes from that it involves highly complex technical knowledge from domain experts, [9], [10]. Hence, Scrum and agile methods attracted attention in PCS development. The main reasons that can be mentioned as their ease of use, the constant communication with the stakeholders and the team and fast development time [10], [15], [16]. In spite of the potential benefits of Scrum, many organizations are reluctant to throw their conventional methods away and jump into agile methods. This to some extent may be attributed some of the issues experienced with the use of Scrum, including significant reduction of documentation, insufficient testing for mission/safety-critical projects, inadequate support for highly stable projects, only successful with talented individuals who favor large degrees of freedom, and inappropriate for large-scale projects [17].

As the discussion above indicates, it is very challenging to specify a PCS at analysis stage. In Scrum-based development this is traditionally done using user stories [18]. The latter nevertheless do present some drawbacks because their narrative alone is not enough to express the constraints related to PCS. This is precisely where BDD could offers complementary representation abilities destined to address user stories limitations. Because of the use of BDD next to user stories narrative, the PCS can be expressed without any other requirements representation artifact (traditionally the Product Variant Master is used for this [3]). Other positive aspects of the use of BDD to further describe user stories are their expression in (structured) natural language making them easy to understand by non-technical stakeholders. They thus present advantages for communication. Finally their dynamic and process oriented nature can be used in a systematic transformation process for the configurator design [19]. All of these elements bring potential positive contributions with the use of BDD and needs to be more formally studied. This paper presents a first attempt towards such a research.

## 2 LITERATURE STUDY

### 2.1 Agile and Scrum

Scrum is an agile software development methodology. The *Agile Manifesto* outlines the values and principles that should be supported by the various agile processes applied in software development. Agile principles emphasize customer satisfaction, change and collaboration between domain experts and developers

[1] Mechanical engineering department, Technical University of Denmark, Denmark, email: sashaf@dtu.dk

[20]. Rubin [21] highlighted that with an agile approach, the team starts by creating a product backlog, which is a prioritized list of the features and other capabilities that need to be developed. Guided by the product backlog, team members address the most important or highest priority items first; priority is based on various factors, but delivered business value is most often the first priority. Scrum is an agile approach for developing innovative products and services [21].

Scrum facilitates cross-team coordination and collaboration [22]. Vlietland et al. (2016) determined that Scrum improves coordination through additional events, such as interteam sprint planning meetings, interteam daily Scrums, interteam product refinements and interteam sprint reviews. A Scrum development life cycle normally consists of short iterations of two to four weeks, an approach that enables swift feedback from software users and related stakeholders regarding the developed solution [19], [23].

## 2.2 Behavior-driven Development

In the movement of agile development, Test-Driven Development (TDD) has been around for a long time and can be traced back to eXtreme Programming practices developed in the late 1990s [24]. In particular, TDD employs so-called acceptance tests as the starting point for the development process to address some of the challenges related to Scrum. Following [25], these TDD relies on two simple principles:

- *Don't write any code until you've written a failing test that demonstrates why you need this code.*
- *Refactor regularly to avoid duplication and keep the code quality high.*

Behavior Driven Development (BDD) has been proposed as a result of the problems that arose with TDD when applying agile software practices [26]. It should be noticed that the language used for describing the tests, i.e. class names and operation names, plays an important role both for writing test cases and for finding bugs in case of a failed test. Inspired by [27], for this purpose BDD uses natural language as a ubiquitous communication mean to describe the acceptance tests by means of scenarios.

The cornerstone of TDD is the idea of writing a unit test before writing the corresponding code. However, BDD is much more than ensuring that every user story has a corresponding set of unit tests; BDD is also about writing specifications, as opposed to tests. In BDD, as an agile software development technique, acceptance tests are written in natural language in order to ensure a common understanding between all members of the project [28]. Consequently, as the first step, the sentences are mapped to actual source code [28].

The shift from TDD to BDD is subtle but significant. Instead of thinking in terms of verification of a unit of code, the focus is on specifying how that code should behave, i.e., what it should do [25]. In order to be sure that of building code that matters, there is a need for specifications that describe what the code should do and how to relate it directly to the business requirements.

Figure 1 shows an example of the BDD flow as it is employed in a specific tool [26].


**Figure 1**. Behavior Driven Development flow

In BDD, as compared to TDD, the task is to write a specification of system behavior that is precise enough for it to be executed as code [29]. More specifically, the whole point of BDD is to ensure that the real business objectives of stakeholders are met by the software we deliver. If stakeholders are not involved, if discussions are not taking place, BDD is not going to work. BDD yields benefits across many important areas such as: (1) Building the right thing, (2) Reducing the risks, (3) Evolving the design [29].

The first Phase in BDD is to understand the business goals and defining features [25]. Vision statement templates make it possible to have a well-defined set of business goals. For a good product vision statement, Moore at al. [30] propose the following contents of a template:

- For (Who will benefit from this product?)
- Who (What do they need?)
- The (What sort of thing are you proposing?)
- That (What makes it so cool?)
- Unlike (What are you competing against?)
- Our product (Why customer prefer your solution?)

Secondly, the features have to be illustrated in natural language to execute the specifications. Consequently, the scenario has the structure [25]:

- Given [context, initial conditions]
- When [event occurs]
- Then [outcome]

There are several studies investigating how to automate all these scenarios such as Cucumber or Jbehave [25].

## 2.3 Requirement artifacts in traditional PCS projects: the Product Variant Master

Generic product structures can be illustrated using the so-called "product variant master" (PVM) notation, which in many cases has functioned as a common language between different product, process and IT experts [3]. Different definitions of the PVM notation have been proposed, and among them the definition of the PVM notation by Haug [11] is one of the most extensive and formalized. A principal example of the PVM technique used to model a toy car solution space is shown in Figure 2. On the left side of a PVM model, part-of-structure is shown, and on the right side, kind-of structure is shown. Classes (typically components and assemblies) are represented by circles and may include attributes and constraints (or rules).

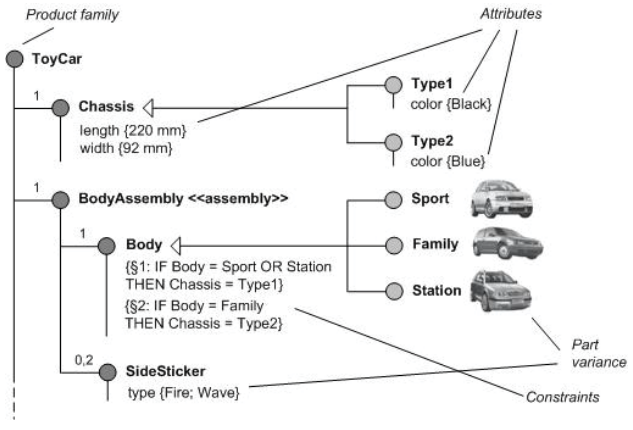


**Figure 2.** Product variant master (ToyCar example)

## 3 PROBLEM STATEMENT

Using Scrum for PCS development introduces both potential benefits and challenges (as for general IT projects in Table 1). One of the challenges is the level of knowledge complexity in PCS that has to be communicated clearly to all Scrum team members in terms of all attributes, constraints and acceptance criteria. An additional challenge from Scrum concerns the lack of visualization (modelling techniques [3]) for product structure. BDD supports Scrum with vision statements [30] to be able to demonstrate the product details and even user interface step by step. Another related challenge is the testing of the PCS as assessing PCSs implies to assess system features with respect to the many possible data and system outputs that might occur when a user is interacting with them. Testing PCSs is an arduous testing activity due to the wide range of user tasks and the different combinations of testing data. BDD, as an add-on of user stories, supports the testers with the detailed defined acceptance criteria.

In short, these concerns bring us three main guidelines for using Scrum methods in PCS projects:

- Formalize user requirements in such a way to provide testability in an ever-changing environment;
- Guarantee consistency between user requirements and their representation in multiple artefacts during development phase; and
- Lay on a validation approach that could be reused to ensure such a consistency for the artefacts along the project.

## 4 RESEARCH METHOD

The aim of this paper is to test the application of BDD in PCS projects in real case projects and gather the data regarding the BDD application. Firstly, we review the literature of BDD to gain deeper understanding of its definition and steps. Secondly, we would apply the findings from literature regarding BDD to the Scrum management in PCS. The authors' ultimate goal is to outline what the contribution of BDD to PCS can be and discuss its importance in promoting the collaboration and communication of knowledge within the organization.

Based on the mentioned challenges in PCS projects, BDD can be an effective solution to improve the definitions of different features and testing the codes in PCS projects next to the user stories. Hence, we posit the following three propositions:

**Proposition 1:** (expressiveness) BDD allows expressing all of the necessary constraints required for documenting a Configurator using Scenarios.

**Proposition 2:** (performance) BDD represents an effective approach for communicating the product specifications when implementing PCS as a replacement for PVM.

**Proposition 3:** (acceptance) BDD represents an effective and practical approach (requirements artifacts comparison) for unit testing of the implemented features in development and testing phases of PCS projects.

We use a qualitative exploratory design based on multiple data sources: requirements artifacts, workshops, interviews and participant observation [31], [32]. The study is ongoing in one case company using Scrum method for developing PCS for more than 4 years. Workshops are conducted to train the team for BDD to be implemented as part of the Scrum. Finally, feedback meetings are held as semi-structured interviews to collect knowledge about the team's satisfaction with BDD. Table 1 summarizes some of the details of the case projects.

**Table 1.** Scrum practice and defined roles and activities in Cases 3 and 4

| *Projects* | *Case 1* | *Case 2* | *Case 3* |
|---|---|---|---|
| Time frame (months) | 11 | 8 | 9 |
| Number of iterations during the project | 10 | 9 | 9 |
| Roles | • Product owner: the stakeholders' manager<br>• Project manager<br>• Scrum master<br>• Development team, including: 1 application manager, 1 project manager, 2 configuration engineers, 2 developers, 1 tester<br>• End users | | |
| Activities | • Backlog grooming<br>• Sprint planning<br>• Sprint backlog<br>• Daily Scrum<br>• Sprint review | | |
| Artefacts (main specifications) | • Product goals and product backlog item (story)<br>• Product backlog and stakeholders' requirements (list of user stories)<br>• Testing (acceptance criteria in user stories) | | |
| Planning approach | • Daily Scrum<br>• Sprint planning<br>• Sprint review<br>• Feedback meetings | | |
| Specific roles of meeting participants | • Same as the project roles, plus:<br>• Product owner: 1<br>• Scrum master: 1<br>• Tester: 1 | | |

## 5 OUTLOOK

It is yet an open question, how much BDD can contribute in the development phase of PCS projects. As PCS projects own their specific challenges, there is a need for further studies to test BDD influence specifically for PCS projects. The results from case studies and interviews will serve to verify or falsify the mentioned hypothesis when the study is accomplished. The structure of a user story presented to the case projects is demonstrated bellow:

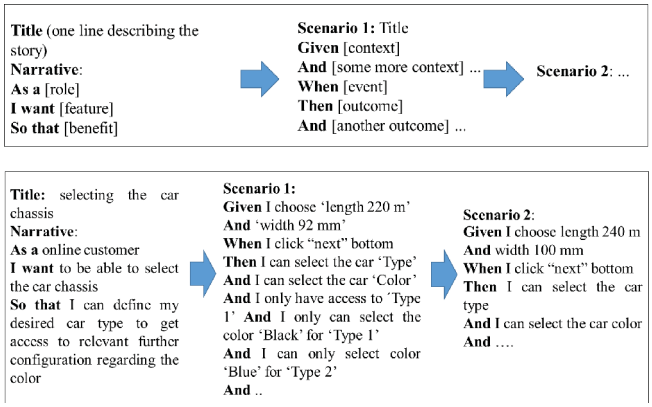


**Figure 3.** BDD structure of a user story presented for PCS projects in the workshops (Car example from Figure 1)

Concerning the adoption next to user stories narrative as a replacement for PVM and the vocabulary proposed in the ontology, an advantage is that requirements and tests in user stories are kept in a natural and high-level language. Based on Figure 3, the team should be able to demonstrate the prototype for user interface. The

evaluation criteria of using BDD approach in the team can be considered as:

1. To support enterprise modeling within agile (user centric) methods
2. To estimate different scenarios, architecture and integration
3. Correct evaluation of available attributes and constraints for the product
4. To support the analysis of requirements communication
5. To analyze the compatibility with business modelling
6. To prototype the desired user interface

## REFERENCES


[1] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-Based Configuration From Research to Business Cases*. Newnes: Morgan Kaufman, 2014.

[2] C. Forza and F. Salvador, *Product information management for mass customization: connecting customer, front-office and back-office for fast and efficient customization*. New York: Palgrave Macmillan, 2007.

[3] L. Hvam, N. H. Mortensen, and J. Riis, *Product customization*. Berlin Heidelberg: Springer, 2008.

[4] A. Felfernig, S. Reiterer, F. Reinfrank, G. Ninaus, and M. Jeran, "Conflict Detection and Diagnosis in Configuration," in *Knowledge-Based Configuration: From Research to Business Cases*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufman, 2014, pp. 73–87.

[5] L. L. Zhang, "Product configuration: a review of the state-of-the-art and future research," *International Journal of Production Research*, vol. 52, no. 21, pp. 6381–6398, Aug. 2014.

[6] A. Haug, L. Hvam, and N. H. Mortensen, "Definition and evaluation of product configurator development strategies," *Computers in Industry*, vol. 63, no. 5, pp. 471–481, Jun. 2012.

[7] S. Shafiee, K. Kristjansdottir, and L. Hvam, "Business cases for product configuration systems," in *7th international conference on mass customization and personalization in Central Europe*, 2016.

[8] S. Shafiee, K. Kristjansdottir, L. Hvam, and C. Forza, "How to scope configuration projects and manage the knowledge they require," *Journal of Knowledge Management*, vol. 22, no. 5, pp. 982–1014, 2018.

[9] A. Felfernig, G. E. Friedrich, and D. Jannach, "UML as domain specific language for the construction of knowledge-based configuration systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 4, pp. 449–469, 2000.

[10] S. Shafiee, L. Hvam, A. Haug, M. Dam, and K. Kristjansdottir, "The documentation of product configuration systems: A framework and an IT solution," *Advanced Engineering Informatics*, vol. 32, pp. 163–175, 2017.

[11] A. Haug, "The illusion of tacit knowledge as the great problem in the development of product configurators," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 26, no. 1, pp. 25–37, Dec. 2010.

[12] C. Forza and F. Salvador, "Product configuration and inter-firm coordination: An innovative solution from a small manufacturing enterprise," *Computers in Industry*, vol. 49, no. 1, pp. 37–46, Sep. 2002.

[13] A. Myrodia, K. Kristjansdottir, S. Shafiee, and L. Hvam, "Product configuration system and its impact on product's life cycle complexity," in *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2016, pp. 670–674.

[14] E. Sandrin, "An empirical study of the external environmental factors influencing the degree of product customization An Empirical Study of the External Environmental Factors Influencing the Degree of Product Customization," no. December 2016, 2017.

[15] B. Selic, "Agile Documentation, Anyone?," *IEEE software*, vol. 26, no. 6, 2009.

[16] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[17] J. Cho, "A Hybrid Software Development Method for Large-Scale Projects: Rational Unified Process with Scrum," *Issues in Information Systems*, vol. 10, no. 2, pp. 340–348, 2009.

[18] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, "Unifying and extending user story models," in *International Conference on Advanced Information Systems Engineering*, 2014, pp. 211–225.

[19] Y. Wautelet, S. Heng, S. Kiv, and M. Kolp, "User-story driven development of multi-agent systems: A process fragment for agile methods," *Computer Languages, Systems and Structures*, vol. 50, pp. 159–176, 2017.

[20] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," *Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pp. 308–313, 2003.

[21] K. S. Rubin, *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[22] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, "The agile requirements refinery: Applying SCRUM principles to software product management," *Information and Software Technology*, vol. 53, no. 1, pp. 58–70, 2011.

[23] J. Vlietland, R. Van Solingen, and H. Van Vliet, "Aligning codependent Scrum teams to enable fast business value delivery: A governance framework and set of intervention actions," *Journal of Systems and Software*, vol. 113, pp. 418–429, 2016.

[24] K. Beck, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[25] S. J. Ferguson, *BDD in Action: Behavior-driven development for the whole software lifecycle*. Manning, 2015.

[26] D. North, "Behavior Modification: The evolution of behavior-driven development," *Better Software*, vol. 8, no. 3, 2006.

[27] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

[28] M. Soeken, R. Wille, and R. Drechsler, "Assisted behavior driven development using natural language processing," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 2012.

[29] S. Nelson-Smith, *Test-Driven Infrastructure with Chef: Bring Behavior-Driven Development to Infrastructure as Code*. O'Reilly Media, Inc., 2013.

[30] G. A. Moore and R. McKenna, *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*. HarperBusiness, 2014.

[31] A. H. Van de Ven, "Nothing is quite so practical as a good theory," *Academy of Management Review*, vol. 14, no. 4, pp. 486–489, 1989.

[32] D. M. McCutcheon and J. R. Meredith, "Conducting case study research in operations management," *Journal of Operations Management*, vol. 11, no. 3, pp. 239–256, Sep. 1993.

# Integrating Semantic Web Technologies and ASP for Product Configuration

Stefan Bischof[1] and Gottfried Schenner[1] and Simon Steyskal[1] and Richard Taupe[1,2]

**Abstract.** Currently there is no single dominating technology for building product configurator systems. While research often focuses on a single technology/paradigm, building an industrial-scale product configurator system will almost always require the combination of various technologies for different aspects of the system (knowledge representation, reasoning, solving, user interface, etc.) This paper demonstrates how to build such a hybrid system and how to integrate various technologies and leverage their respective strengths.

Due to the increasing popularity of the industrial knowledge graph we utilize Semantic Web technologies (RDF, OWL and the Shapes Constraint Language (SHACL)) for knowledge representation and integrate it with Answer Set Programming (ASP), a well-established solving paradigm for product configuration.

## 1 INTRODUCTION

The way large organizations manage their data is subject to trends. Currently the (industrial) knowledge graph is gaining popularity [13]. The term "knowledge graph" was coined by Google [20] in the context of search engines. This was adopted by the industry to describe a system that manages the data of a company with a graph-based formalism like RDF and combines it with reasoning (e.g. OWL) and machine learning [3]. Having all internal data of the company accessible in one format solves the problem of isolated data silos often found in large corporations and allows the combination of internal data with external data e.g. from the Linked Open Data cloud [4].

Knowledge graphs rely heavily on methods and technologies developed by the Semantic Web community. Given the current acceptance in the industry for these technologies it is a good time for the product configuration community to revisit these technologies. In this paper we demonstrate how to use Semantic Web technologies for product configuration and how to integrate them with established solving technologies for product configuration like Answer Set Programming.

Why is the use of open standards and technologies like the Semantic Web technology stack so important? In our experience using a configurator with a vendor-specific language for specifying product configuration problems is the equivalent of a data silo in data management. It is very hard to switch from one configurator vendor to another, if both systems use their own proprietary specification language. Also in the product configuration community there is currently no standard way to specify product configuration problems although the topic of ontologies and product configuration is over 20 years old (cf. [21]) and pre-dates the Semantic Web.

In Section 2 we describe the technologies used for this paper. In Section 3 we show how to define a product configurator knowledge base with RDF and SHACL, that allows checking of constraints and interactive solving.

For solving product configuration problems, RDF and SHACL are combined with Answer Set Programming in Section 4. In Section 5 we illustrate how reasoning with OWL can help to integrate the product configuration solutions into the knowledge graph and facilitate reuse of ontologies.

In Section 6 we give a brief overview of the systems used for the examples and in Section 7 we conclude.

[1] Siemens AG Österreich, Corporate Technology, Vienna, Austria
bischof.stefan@siemens.com, gottfried.schenner@siemens.com, simon.steyskal@siemens.com, richard.taupe@siemens.com
Author names are given in alphabetical order.
[2] Alpen-Adria-Universität, Klagenfurt, Austria

## 2 PRELIMINARIES

The proposed approach builds heavily on Semantic Web standards and technologies. Instance data is represented as RDF triples, domain models are mapped to domain-dependent ontologies/vocabularies and queries are formulated in SPARQL [19].

### 2.1 RDF+SHACL

The Resource Description Framework (RDF) [6] is a both human-readable and machine-processable framework for describing and representing information about resources. In RDF every resource is identified by an IRI, and information about resources is represented in form of triples with $I \cup B \times I \times I \cup B \cup L$, where $I$, $B$, $L$ denote IRIs, blank nodes (nodes that do not have a corresponding IRI and which are mainly used to describe special types of resources without explicitly naming them) and RDF literals (e.g., strings, integers, etc.) respectively.

The Shapes Constraint Language (SHACL) [9] – a W3C Recommendation since 2017 – is a language for validating RDF graphs against a set of constraints. Its validation process is built around the notion of *Data Graphs* (RDF graphs that contain the data that has to be validated), and *Shapes Graphs* (RDF graphs containing shape definitions and other information that is used to perform the validation of the *Data Graphs*).

## 2.2 Ontologies

The *OWL 2 Web Ontology Language* (OWL) [12] is a declarative knowledge representation formalism standardized by the W3C in 2012. OWL is a language to represent *ontologies* and is based on description logics.

An ontology describes things (*individuals*), sets of individuals (*classes*), relations between individuals (*object properties*) and attributes of individuals (*data properties*). Based on these, OWL provides class and property constructors to define complex classes and properties, e.g., intersection or union of classes. Eventually, with OWL *axioms*, we can define how classes (and properties) are related to each other, e.g., subclasses, equivalent classes, or disjoint classes.

OWL ontologies can be serialized in one of several syntaxes. In this paper we use Turtle [2] syntax for serializing both OWL and SHACL.

## 2.3 Validation vs. Inference

In the Semantic Web, the tasks of (*i*) constraint validation and (*ii*) reasoning are grounded on different semantics. While the latter usually operates under the *Open World Assumption (OWA)* (i.e., a statement cannot be assumed to be false if it cannot be proven to be true [16]) and the *Non-Unique Name Assumption (nUNA)* (i.e., the same resource can have multiple names), validation adheres to the *Closed World Assumption (CWA)* (i.e., a statement is assumed to be false if it cannot be proven to be true) and requires that different names identify different objects (i.e., it makes the *Unique Name Assumption (UNA)*).

Those differences are illustrated in Figure 1 where OWL is used for specifying a cardinality constraint for individuals of type `:ElementA` allowing them to refer to exactly one module of type `:ModuleI` only.

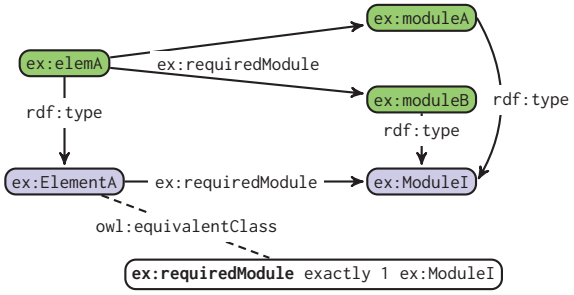


**Figure 1.** `ex:elemA` violates its cardinality constraint only under UNA/CWA, but not under nUNA/OWA where it could be inferred that `ex:moduleA` and `ex:moduleB` represent the same resource, thus not violating the cardinality constraint.

Class `ex:ElementA` is defined to be equal to an anonymous class having exactly one `ex:ModuleI` associated via `ex:requiredModule`, hence every individual of type `ex:ElementA` is only allowed to refer to one specific module. Even though `ex:elemA` is violating its cardinality constraint (because it is associated to more than one module via `ex:requiredModule`) an OWL reasoner would not be able to detect a violation, but instead infers that `ex:moduleA` and `ex:moduleB` are representing the same thing.

## 2.4 ASP

Answer Set Programming (ASP) is a declarative programming paradigm. Instead of specifying how to find a solution to a problem in terms of an imperative algorithm, in ASP the problem itself is specified in the form of a logic program. We restrict our introduction to ASP to core concepts needed to understand this paper and refer to [5,10,11] for more details.

A program $P$ is a finite set of rules of the form

$$\mathtt{h}\ \texttt{:-}\ \mathtt{b}_1, \ldots, \mathtt{b}_m, \ \texttt{not}\ \mathtt{b}_{m+1}, \ \ldots, \ \texttt{not}\ \mathtt{b}_n.$$

where $\mathtt{h}$ and $\mathtt{b}_1, \ldots, \mathtt{b}_m$ are positive literals (i.e. atoms) and $\texttt{not}\ \mathtt{b}_{m+1}, \ldots, \texttt{not}\ \mathtt{b}_n$ are negative literals. An atom is either a classical atom or a cardinality atom. A classical atom is an expression $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ where $\mathtt{p}$ is an $n$-ary predicate and $\mathtt{t}_1, \ldots, \mathtt{t}_n$ are terms. A term is either a variable (whose name starts with an upper-case character or an underscore) or a constant (which can be a number or a string). A literal is either an atom $\alpha$ or its default negation $\texttt{not}\ \alpha$. Default negation refers to the absence of information, i.e. an atom is assumed to be false as long as it is not proven to be true. Thus, ASP makes the Closed World Assumption.

A *cardinality atom* is of the form

$$\mathtt{l} \prec_1 \{\ \mathtt{a}_1 : \mathtt{l}_{1_1}, \ldots, \mathtt{l}_{1_m}\ ;\ \ldots;\ \mathtt{a}_n : \mathtt{l}_{n_1}, \ldots, \mathtt{l}_{n_o}\} \prec_u \mathtt{u}$$

where each structure $\mathtt{a}_i : \mathtt{l}_{i_1}, \ldots, \mathtt{l}_{i_m}$ is a *conditional literal* in which $\mathtt{a}_i$ (the head of the conditional literal) and all $\mathtt{l}_{i_j}$ are classical literals, and $\mathtt{l}$ and $\mathtt{u}$ are terms representing non-negative integers indicating lower and upper bound. If one or both of the bounds are not given, their defaults are used, which are 0 for $\mathtt{l}$ (and $\leq$ for $\prec_1$) and $\infty$ for $\mathtt{u}$ (and $\leq$ for $\prec_u$).

$H(r) = \{\mathtt{h}\}$ is called the *head* of the rule, and $B(r) = \{\mathtt{b}_1, \ldots, \mathtt{b}_m, \texttt{not}\ \mathtt{b}_{m+1}, \ldots, \texttt{not}\ \mathtt{b}_n\}$ is called the body of the rule. A rule with empty head is a *constraint* and is used to filter out invalid solutions. A rule with empty body is called *fact*, its head holds unconditionally in a satisfiable program.

There are several ways to define the semantics of an answer-set program, i.e. to define the set of answer sets $AS(P)$ of an answer-set program $P$. An overview is provided by [11]. Informally, an answer set $A$ of a program $P$ is a subset-minimal model of $P$ (i.e. a set of atoms interpreted as *true*) which satisfies the following conditions: All rules in $P$ are satisfied by $A$; and all atoms in $A$ are "derivable" by rules in $P$. A rule is satisfied if its head is satisfied or its body is not. A cardinality atom is satisfied if $l \prec_1 |C| \prec_u u$ holds, where $C$ is the set of head atoms in the cardinality literal whose conditions (e.g. $\mathtt{l}_{i_1}, \ldots, \mathtt{l}_{i_m}$ for $\mathtt{a}_i$) are satisfied and which are satisfied themselves.

Most ASP systems split the solving process into grounding and solving. The former part produces the grounding of a program, i.e. its variable-free equivalent. Thereby, the variables in each rule of the program are substituted by constants. The latter part then solves this propositional encoding.

To briefly illustrate ASP by means of a small example, consider the following program:

```
triple("ex:ElementA", "rdfs:subClassOf", "ex:Element").
rdfs_subClassOf(Sub,Sup) :-
 triple(Sub, "rdfs:subClassOf", Sup).
class(Sub) :- rdfs_subClassOf(Sub,Sup).
class(Sup) :- rdfs_subClassOf(Sub,Sup).
n_element_types(N) :-
 N = { rdfs_subClassOf(Sub,"ex:Element") : class(Sub) }.
```

This program contains one fact representing an RDF triple[3] and two rules that derive new atoms from it. The single answer set of this program is:

```
{
 triple("ex:ElementA","rdfs:subClassOf","ex:Element"),
 rdfs_subClassOf("ex:ElementA","ex:Element"),
 class("ex:ElementA"), class("ex:Element"),
 n_element_types(1)
}
```

## 3 PRODUCT CONFIGURATION WITH SHACL

The following describes the product configuration terminology from [8] adapted to RDF + SHACL.

**Definition 1** *(Configuration Model) The (SHACL) configuration model CONFMODEL = (SHAPEGRAPH, RDF) consists of a SHACL shapes graph and an ontology/RDFS schema defining all the used classes and properties in the configuration model.*

**Definition 2** *(User requirements) The (SHACL) user requirements USERREQ = (SHACLCONSTRAINTS, SUBGRAPH) consists of additional SHACL constraints and an initial RDF subgraph.*

**Definition 3** *(Configuration Task) The configuration task CONFIGTASK = (CONFMODEL, USERREQ) represents the input of one concrete configuration problem.*

**Definition 4** *(Configuration) A configuration (solution) of a configuration task CONFIGTASK is an RDF graph, which satisfies the SHACL constraints of CONFMODEL and USERREQ and contains SUBGRAPH of USERREQ as a sub graph.*

The configuration model defines the constraints that all configurations must satisfy. The ontology is used to identify the classes and properties relevant for the current configuration task. In a large knowledge graph (RDF store/graph database) there can be thousands of concepts and properties and typically only a small subset is relevant for product configuration. Even within the classes relevant for product configuration not all will be relevant in the current configuration task as the configuration task of large artefacts is typically split into smaller configuration tasks like hardware configuration and software configuration. Although the tasks are separated, the resulting configurations will reside in the same knowledge graph and will be linked, e.g., a hardware component will be related to the required software driver.

In the following we will illustrate the product configuration concepts by means of an example.

### 3.1 Running example

For ease of comparison we use the example of [18] as a running example. This is an abstract example of a typical hardware configuration problem found in industry with some key features like a component taxonomy, cardinality restrictions for relations between component types, etc.

In our example domain there may be different types of elements, which are controlled by hardware modules. Each hardware module must be in a frame and a frame must be mounted on a rack. More specifically, the constraints of the domain are:

- There are four disjoint types of elements (ElementA-ElementD).
- There are five disjoint types of modules (ModuleI-ModuleV).
- There are two disjoint types of racks (RackSingle, RackDouble).
- An ElementA/B/C/D requires exactly one/two/three/four ModuleI/II/III/IV respectively, i.e., an ElementB must have only and exactly two ModuleIIs associated via the requiredModule property.
- A ModuleV cannot have an element, all other modules must be required by an element (via the requiredModule property).
- A RackSingle must contain exactly four frames, a RackDouble must contain exactly eight frames.
- A frame must be mounted on a rack.
- A frame can contain up to six modules.
- A module must be mounted on a frame.
- Whenever a frame contains a module of type ModuleII, it must also contain one of type ModuleV.

The ontology of our running example is illustrated in Fig. 2.

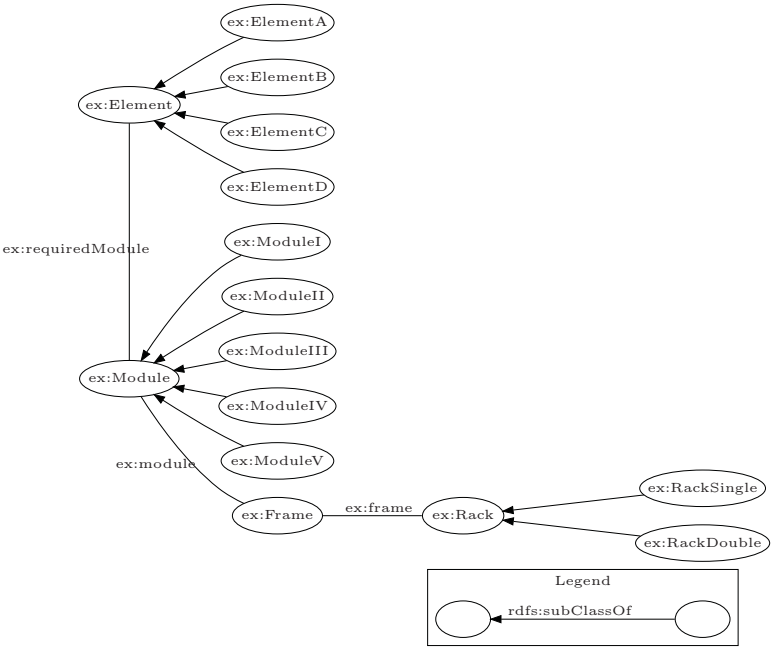


**Figure 2.** Racks ontology RDF graph

The following shows an excerpt of the ontology in turtle format:

```
ex:ElementA rdfs:subClassOf ex:Element .
ex:ElementB rdfs:subClassOf ex:Element .
ex:ElementC rdfs:subClassOf ex:Element .
ex:ElementD rdfs:subClassOf ex:Element .
```

[3] For conciseness, we use RDF prefixes also when we present ASP encodings in this paper, even though the real encodings contain full IRIs. For example, the string `"rdfs:subClassOf"` stands for `"http://www.w3.org/2000/01/rdf-schema#subClassOf"`.

```
ex:ModuleI rdfs:subClassOf ex:Module .
ex:ModuleII rdfs:subClassOf ex:Module .
ex:ModuleIII rdfs:subClassOf ex:Module .
ex:ModuleIV rdfs:subClassOf ex:Module .
ex:ModuleV rdfs:subClassOf ex:Module .

# defines the objectproperty relating elements
# to their required modules
# the cardinality constraints for the objectproperty
# are defined with SHACL constraints
:requiredModule rdf:type owl:ObjectProperty ;
  rdfs:domain :Element ;
  rdfs:range :Module .
```

## 3.2 SHACL constraints

In the subsequent paragraphs, we illustrate some types of constraints supported by SHACL that are relevant in the configuration domain and refer to [9] for a full account of SHACL constraints.

**Cardinality constraints** A typical class of constraints for industrial product configuration problems are cardinality constraints. Whereas in customer product configuration problems simple yes/no or mandatory/optional constraints between components/features are often sufficient, complex cardinality constraints are common in industrial configuration problems, e.g., in our example each element class requires a different number and type of attached module. In SHACL these restrictions can be expressed with qualified cardinality constraints. For example the constraint that each instance of `ex:ElementB` requires exactly 2 `ex:ModuleII` can be expressed like this:

```
ex:ElementBRequiredModuleShape
  a sh:NodeShape ;
  sh:targetClass ex:ElementB ;
  sh:property [
    sh:path ex:requiredModule ;
    sh:minCount 2 ;
    sh:maxCount 2 ;
    sh:class ex:ModuleII ;
  ] .
```

Cardinality constraints are a powerful mechanism of product configuration models. With them it is possible to express mandatory/optional, requires, and part/subpart relationships. Almost all the constraints of our running example can be expressed with cardinality constraints.

**Completeness of Taxonomy** Typically in a configuration model it is required that every object of a configuration must be an instance of a leaf class in the taxonomy, e.g., if a configuration contains a rack, it must be known whether it is a RackSingle or a RackDouble. In SHACL this constraint can be expressed as:

```
# subclass inheritance disjoint and complete
ex:RackSubclassShape
  a sh:NodeShape ;
  sh:targetClass ex:Rack ;
  sh:message
        "A Rack must be either of
        type ex:RackSingle or ex:RackDouble" ;
  sh:xone (
     [  sh:class ex:RackSingle ]
     [  sh:class ex:RackDouble ]
  ) .
```

The reason for requiring completeness is that a configuration should be a complete description of all components the configured artefact contains. If the exact type of a component is not known, the solution may be underspecified.

There are some scenarios like ETO (engineer to order) where this restriction does not apply. In these cases some parts of the configured artefact are deliberately left unspecified because it is the task of the engineering department to come up with a solution, e.g., in our example we could add as a SHACL constraint that every rack needs a power supply but let the exact type of power supply unspecified. It is then the task of the engineering department to find a suitable power supply for the given configuration.

## 3.3 Checking constraints

Once we have defined the SHACL configuration model we can use it to check if a given RDF graph is a valid configuration. For example checking the RDF graph consisting of the single triple `ex:EB a ex:ElementB .` against previously introduced SHACL constraints produces the following validation result:

```
[
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:sourceConstraintComponent
            sh:MinCountConstraintComponent ;
    sh:sourceShape _:n236 ;
    sh:focusNode ex:EB ;
    sh:resultPath ex:requiredModule ;
    sh:resultMessage "Less than 2 values" ;
] .
```

## 3.4 Interactive solving

In an interactive setting the user of the configurator will start with a non-empty subgraph and the system will indicate all currently violated SHACL constraints. The following shows a (fictional) session between user and configurator (conf), where the user asserts additional triples and the configurator reports constraint violations:

```
user: ex:E1 a ex:Element .
conf: ex:E1 must be oneof {ElementA, ..., ElementD}
user: ex:E1 a ex:ElementA .
conf: ex:E1 requires one ModuleI
user: ex:E1 ex:requiredModule ex:M1 .
user: ex:M1 a ex:ModuleI .
conf: ex:M1 requires a Frame
user: ex:F1 ex:module ex:M1 .
user: ex:F1 a ex:Frame .
conf: ex:F1 requires a Rack
user: ex:R1 ex:frame ex:F1 .
user: ex:R1 a ex:RackSingle .
conf: ex:R1 requires 4 frames
user: ex:R1 ex:frame ex:F2; ex:F3; ex:F4 .
user: ex:F2 a ex:Frame .
user: ex:F3 a ex:Frame .
user: ex:F4 a ex:Frame .
conf: No constraints violated
```

The resulting configuration of this configuration task is shown in Fig. 3 .

Although that kind of interaction might be sufficient for a domain expert, it is clear that the interactive configuration task shown above can be improved. There are only two points in the configuration task where a decision by the user is required. The first decision is to create an ElementA and the second decision is to choose between an RackSingle and a RackDouble. All other statements could be automatically derived by a reasoner.

Some simple statements can be derived with an RDFS reasoner. For example, `ex:F2 a ex:Frame .` follows from the

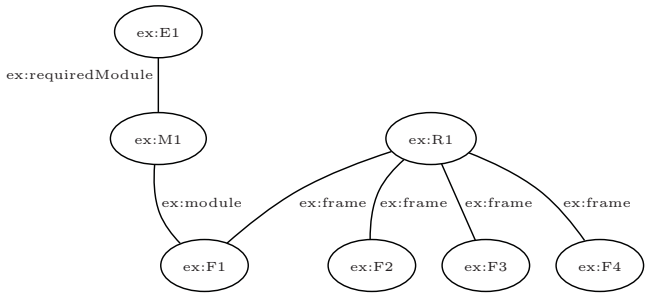


**Figure 3.** Racks configuration example

fact that `ex:F2` is an object in an `ex:frame` property and this property is defined to have the range `ex:Frame`.

Other SHACL constraints cannot be repaired by a typical Semantic Web reasoner because they require the creation of new entities. For these cases SHACL rules can be applied. The following example illustrates how SHACL rules could be utilized to "repair" a SHACL constraint violation. This rule creates frames for a rack until the required number of frames was created.

```
ex:RackRuleShape
  a sh:NodeShape ;
  sh:targetClass ex:RackSingle ;
  sh:rule [
    a sh:SPARQLRule ;
    sh:prefixes ex: ;
    sh:construct """
      PREFIX ex: <http://example.org/confws2018#>
      CONSTRUCT {
        this ex:frame _:new .
        _:new a ex:Frame .
      }
      WHERE { }
    """ ;
    sh:condition [ sh:not ex:RackSingleFrameShape ] ;
  ] .
```

# 4 SOLVING SHACL CONFIGURATIONS WITH ASP

Unfortunately, SHACL rules are a relatively new concept and there are no rule engines that support backtracking. Therefore one must revert to more established solving techniques for product configuration. For this paper we decided to use ASP.

As a basic proof of concept, we demonstrate the generic translation of SHACL constraints to ASP for concepts needed in the example illustrated in Section 3.1. As such, this demonstration includes only a subset of the SHACL Core Validators [9, Appendix D]. Most of the other validators can be expressed in ASP as well[4].

## 4.1 General Concepts

We represent RDF triples in ASP as instances of the ternary predicate `triple`, whose arguments are `Subject`, `Predicate`, and `Object`. For convenience, triples of certain RDF predicates are mapped to smaller ASP atoms by a set of projection rules:

```
rdfs_subClassOf(Sub,Sup) :-
 triple(Sub,"rdfs:subClassOf",Sup).
a(Instance,Class) :-
 triple(Instance,"rdf:type",Class).
sh_target_class(S, C) :- triple(S, "sh:targetClass", C).
sh_node_property(NS, PS) :- triple(NS, "sh:property", PS).
sh_property_shape(PS) :- triple(NS, "sh:property", PS).
sh_property_shape(PS) :-
 triple(PS, "rdf:type", "sh:PropertyShape").
sh_property_minCount(PS, Min) :-
 triple(PS, "sh:minCount", Min).
sh_property_maxCount(PS, Max) :-
 triple(PS, "sh:maxCount", Max).
sh_xone(S, List) :-
 triple(S, "sh:xone", List).
```

To enable the reasoning component to not only rule out invalid solutions but also explain inconsistencies, we use `shacl_constraint_violated` atoms in the head of constraints together with SHACL messages to encode explanations in the form of `shacl_constraint_violation_message` atoms in answer sets:

```
sh_message(S,Msg) :-
  triple(S,"sh:message",Msg).
shacl_constraint_violation_message(I,S,Msg) :-
  shacl_constraint_violated(I,S), sh_message(S,Msg).
```

## 4.2 Property Shapes

Property shapes specify constraints that need to be fulfilled by nodes that are reached on a SHACL property path, which can be defined in various ways [9, Section 2.3]. These constraints and paths need to be mapped to ASP concepts to enable mapping of ASP constraints to the intended targets of a SHACL property. A subset of these paths is handled by the following encoding:

```
sh_property_path(PS, P) :-
 triple(PS, "sh:path", P), not sh_path_is_inverse(P).
sh_path_is_inverse(P) :- triple(P, "sh:inversePath", _).
sh_property_path_inv(PS, InvP) :-
 triple(PS, "sh:path", P),
 triple(P, "sh:inversePath", InvP).

shacl_property_target(PS, Trgt) :-
 sh_property_target_inst(PS, Inst, Trgt).
sh_property_target_inst(PS, Inst, Trgt) :-
 sh_property_shape(PS), sh_node_property(NS, PS),
 sh_property_path(PS, P), sh_targetClass(NS, Class),
 a(Inst, Class), triple(Inst, P, Trgt).
sh_property_target_inst(PS, Inst, Trgt) :-
 sh_property_shape(PS), sh_node_property(NS, PS),
 sh_property_path_inv(PS, InvP), sh_targetClass(NS, Class),
 a(Inst, Class), triple(Trgt, InvP, Inst).
```

## 4.3 Class Membership

As an example for a simple SHACL constraint, we provide an encoding for the condition that a value node must be a SHACL instance of a given type (cf. [9, Section 4.1.1]):

```
shape_constraint(S, class, C) :-
 triple(S, "sh:class", C).
shape_constraint_satisfied_inst(I, S, class, C) :-
 shape_constraint(S, class, C), a(I, C).
```

## 4.4 Cardinality Constraints

Based on the encoding fragments presented so far we are now able to encode detection of cardinality constraint violations (cf. [9, Section 4.2]):

```
shacl_constraint_violated(I,PS) :-
 sh_property_shape(PS), sh_node_property(NS,PS),
 sh_target_class(NS,C), a(I,C), sh_property_minCount(PS,Min),
 not Min { sh_property_target_inst(PS,I,T) }.
shacl_constraint_violated(I,PS) :-
 sh_property_shape(PS), sh_node_property(NS,PS),
 sh_target_class(NS,C), a(I,C), sh_property_maxCount(PS,Max),
 not { sh_property_target_inst(PS,I,T) } Max.
```

[4] Validators that rely on string operations or data types are difficult to encode in ASP.

### 4.5 Logic Constraints

Logic constraints like xone (cf. [9, Section 4.6]) can also be mapped to cardinality constraints:

```
sh_xone_inst(Inst, List) :- sh_xone(S, List),
 sh_target_class(S, Class), a(Inst, Class).
shacl_constraint_violated(Inst, S) :-
 sh_xone(S, List), sh_xone_inst(Inst, List),
 not 1 {
  shape_constraint_satisfied_inst(
   Inst, List, Constraint, Value
  ) : shape_constraint(List, Constraint, Value)
 } 1.
```

### 4.6 Solving

Given a translation of the SHACL constraints into ASP we can check if the given RDF graph is a valid configuration, i.e., ASP acts as an implementation of the SHACL validator. To enable solving, an additional generative program is needed. This generative program must be capable of enumerating all possible solutions within a certain scope. The generative program together with the translated SHACL constraints enables us to find a valid configuration, if one exists within the given scope. The following listing shows a generative program capable of enumerating all configurations consisting of racks and frames. The scope is controlled by blank nodes (e.g., `_:b1`). Every blank node can become a new component within the configuration.

```
bnode("_:b1").
bnode("_:b2").
bnode("_:b3").
bnode("_:b4").
bnode("_:b5").
bnode("_:b6").

configobj(O) :-
    triple(O,"a",C),
    configclass(C).

0 { triple(BNODE, "a", "ex:RackSingle") } 1 :-
    bnode(BNODE).
0 { triple(BNODE, "a", "ex:RackDouble") } 1 :-
    bnode(BNODE).
0 { triple(BNODE, "a", "ex:Frame") } 1 :-
    bnode(BNODE).

0 { triple(O1, "ex:frame", O2)} 1 :-
    configobject(O1),
    configobject(O2).

% answer set found
triple("_:b1","a","ex:Rack").
triple("_:b1","a","ex:RackSingle").
triple("_:b2","a","ex:Frame").
triple("_:b3","a","ex:Frame").
triple("_:b4","a","ex:Frame").
triple("_:b5","a","ex:Frame").
triple("_:b1","ex:frame","ex:b2").
triple("_:b1","ex:frame","ex:b3").
triple("_:b1","ex:frame","ex:b4").
triple("_:b1","ex:frame","ex:b5").
```

After an answer set has been found due to the triple notation it can be directly translated back to an RDF graph. This RDF graph is a solution of the configuration task and satisfies all SHACL constraints.

## 5 COMBINING SHACL/RDF WITH OWL REASONING

So far we have concentrated on closed world reasoning and the unique name assumption for checking and finding configurations. In practice there are applications for CWA/OWA and UNA/nUNA. As a use case for open world reasoning we will demonstrate how to reuse an existing ontology for our running example.

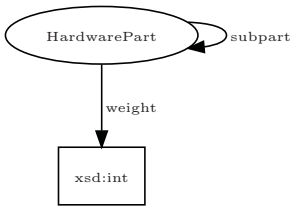


**Figure 4.** Hardware part Ontology (:hw)

### 5.1 Semantic differences of SHACL and OWL reasoning

Consider the constraint that a single rack has four frames. Due to closed world reasoning the SHACL constraint is violated in the example below. For a OWL reasoner the example is consistent, because OWL adheres to the open world assumption and thus an OWL reasoner would not be able to decide if there is another frame.

```
ex:R1 a ex:RackSingle .
ex:R1 ex:frame ex:F1 .
ex:R1 ex:frame ex:F2 .
ex:R1 ex:frame ex:F3 .
```

In the next example again a SHACL constraint will be violated, because there are 5 frames associated with a Rack and under the Unique Name Assumption they are all considered different entities. In OWL this example is consistent, because the unique name assumption is not applied and there is no statement indicating that, e.g., `ex:F1` and `ex:F2` are different entities.

```
ex:R1 a ex:RackSingle .
ex:R1 ex:frame ex:F1 .
ex:R1 ex:frame ex:F2 .
ex:R1 ex:frame ex:F3 .
ex:R1 ex:frame ex:F4 .
ex:R1 ex:frame other:FA .
```

### 5.2 Reusing ontologies with OWL reasoning

Modeling subpart relations can be surprisingly complicated (cf. [7]), but is necessary in many knowledge representation systems. For presentational reasons we opt for a simplistic ontology: there exist hardware parts which can have subparts which are also hardware parts. The *subpart* relation is intentionally not defined as transitive and thus expresses only direct subparts. Additionally each hardware part must have a weight (see Fig. 4). Having hardware parts modeled along this ontology, allows us to (recursively) compute the *total weight* of a hardware part by adding the weight of the part itself with the sum of the *total weights* of all (direct) subparts. However, recursion is not needed, and so we will instead compute the total weight of a hardware part by adding its own weight to the weight to all (transitively reachable) subparts. The following SPARQL query performs this computation for all hardware parts (naturally the variable `?part` in the `WHERE` clause could be replaced by the URI of some hardware part):

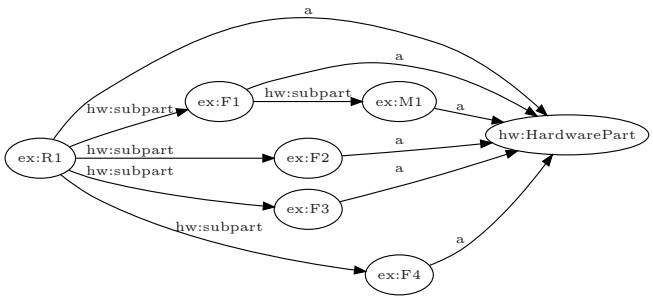


**Figure 5.** Configuration example mapped to hardware part ontology

```
SELECT ?part (SUM(?weight) as ?totalweight)
WHERE {
  ?part p:subpart*/p:weight ?weight .
} GROUP BY ?part
```

But before we can do that, we have to map the configuration created previously to this Hardware Part ontology. The mapping can be expressed by SHACL rules or a *mapping ontology*. Taking the latter approach we can hold off on the decision for a concrete implementation strategy. In our example we only have to define the object properties `ex:frame` and `ex:module` as subproperties of `p:subpart` in the mapping ontology. Taking all the ontologies (example, subpart, and mapping) and the racks configuration data into account, an OWL reasoner would (also) entail the subpart relationships depicted in Fig. 5. Depending on the OWL reasoner implementation these entailments are materialised or only inferred for relevant queries. Assuming the weights of the hardware parts themselves are also available (for example through a similar mapping to a product catalogue) we now have all the necessary parts to compute the total weight of all the hardware parts with the above SPARQL query.

Alternatively we could use a similar query to materialise the total weights in the RDF graph:

```
INSERT { ?part p:totalweight ?totalweight }
WHERE { {
  SELECT ?part (SUM(?weight) as ?totalweight)
  WHERE {
    ?part p:subpart*/p:weight ?weight .
  } GROUP BY ?part
} }
```

Using this mapping approach we can transparently map our instance data with relevant data from other (legacy) information systems via a knowledge graph and thus create a more complete knowledge base. We therefore are able to reuse ontologies like the Hardware Part ontology and the associated SPARQL queries in a modular manner.

## 6 EVALUATION

All the example RDF files, ontologies, ASP and Java programs are available in our open-source git repository[5]. For manipulating RDF the Apache Jena library version 3.5.0[6] was used. For checking SHACL constraints we relied on the TopBraid SHACL API version 1.1.0[7]. The SHACL examples can also be checked online interactively by using the SHACL playground[8]. The SHACL rule example has been implemented in Java. The translation of the SHACL example to ASP is also implemented in Java. For running the ASP programs we used the ASP solver clingo version 5.2.0[9]. The OWL ontologies were edited with the Protégé ontology editor[10] version 5.2. The classification example of 5 was verified with the integrated HermiT reasoner. The example was also executed with StarDog version 5.3.0[11] by using a SPARQL query with activated OWL reasoning. StarDog is a knowledge graph platform for the enterprise.

## 7 CONCLUSION

We have used Semantic Web technologies in the past, but only for specific purposes like data integration [17]. Product configuration knowledge bases require closed world reasoning and the default open world reasoning of OWL made it cumbersome to be used for that task in our experience. We found that SHACL closes this gap and have demonstrated in this paper how to define a configurator knowledge base just with RDF+SHACL. Together with constraint validation such a system can be the basis of a simple interactive configurator. SHACL rules can enhance the user experience by deriving additional knowledge. We do not expect SHACL to be *the* language for specifying product configuration problems in combination with ontologies but it is a step into the right direction.

Because of the lack of backtracking SHACL rule engines, SHACL rules can currently not be used to solve configurations except for trivial examples. A solver for product configuration problems finds a model for the product configuration model. This is not a reasoning task supported by a typical Semantic Web reasoner. The main task for a Semantic Web reasoner is classification and determining consistency[12].

Therefore we resorted to ASP for solving the configuration problem. To demonstrate the feasibility of the approach we translated the subset of SHACL required to solve our simple example domain. We will continue to evaluate our approach on more sophisticated domains e.g. by adding arithmetic constraints. Because the semantic of SHACL can be defined by SPARQL and the expressive power of SPARQL [1] is sufficient for typical product configuration domains we expect no conceptual difficulties.

The main challenge will be the automatic translation of the SHACL constraints into ASP. In the future we want to adopt a more generic approach by building on the work already done for SPARQL and ASP [14, 15]. Of course such a translational approach is not restricted to ASP. The same could be applied to SAT, CSP or any other solving paradigm for product configuration.

An important topic for the future will be how to identify the relevant information for product configuration in the knowledge graph. This includes how to (semi-)automatically identify the parts that are currently available for configuring a

[5] https://github.com/siemens/ProductConfigurationWithSHACL
[6] https://jena.apache.org/
[7] https://github.com/TopQuadrant/shacl

[8] http://shacl.org/playground/
[9] https://potassco.org/
[10] https://protege.stanford.edu/
[11] https://www.stardog.com/
[12] A Semantic Web reasoner might eventually produce a model internally for proving consistency, but this is normally not available to the caller of the reasoner.

product, how to relate the product configuration ontologies to other relevant ontologies for product line management, enterprise resource planning etc.

## ACKNOWLEDGEMENTS

This work was partially conducted within the scope of DynaCon (FFG-PNr.: 861263), which is funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between November 2017 and April 2020. More information: https://iktderzukunft.at/en/

## REFERENCES

[1] Renzo Angles and Claudio Gutierrez, 'The expressive power of sparql', in *The Semantic Web - ISWC 2008*, eds., Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, pp. 114–129, Berlin, Heidelberg, (2008). Springer Berlin Heidelberg.

[2] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers, 'RDF 1.1 Turtle – terse RDF triple language', Recommendation, W3C, (February 2014).

[3] Luigi Bellomarini, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger, 'Swift logic for big data and knowledge graphs', in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, ed., Carles Sierra, pp. 2–10. ijcai.org, (2017).

[4] Christian Bizer, Tom Heath, and Tim Berners-Lee, 'Linked data - the story so far', *Int. J. Semantic Web Inf. Syst.*, **5**(3), 1–22, (2009).

[5] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński, 'Answer set programming at a glance', *Communications of the ACM*, **54**(12), 92–103, (2011).

[6] Richard Cyganiak, David Wood, and Markus Lanthaler, 'Resource Description Framework (RDF)', Recommendation, W3C, (February 2014). Available at https://www.w3.org/TR/rdf11-concepts/.

[7] Mariano Fernández-López, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa, 'Selecting and customizing a mereology ontology for its reuse in a pharmaceutical product ontology', in *Proceedings of the 2008 Conference on Formal Ontology in Information Systems: Proceedings of the Fifth International Conference (FOIS 2008)*, pp. 181–194, Amsterdam, The Netherlands, The Netherlands, (2008). IOS Press.

[8] Lothar Hotz, Alexander Felfernig, Markus Stumptner, Anna Ryabokon, Claire Bagley, and Katharina Wolter, 'Chapter 6 - configuration knowledge representation and reasoning', in *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 41–72, Morgan Kaufmann, Boston, (2014).

[9] Holger Knublauch and Dimitris Kontokostas, 'Shapes Constraint Language (SHACL)', Recommendation, W3C, (July 2017). Available at https://www.w3.org/TR/shacl/.

[10] Vladimir Lifschitz, 'What Is Answer Set Programming?', in *Twenty-Third AAAI Conference on Artificial Intelligence*, (2008).

[11] Vladimir Lifschitz, 'Thirteen Definitions of a Stable Model', in *Fields of Logic and Computation*, eds., Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, volume 6300 of *Lecture Notes in Computer Science*, 488–503, Springer, Berlin, Heidelberg, (2010).

[12] W3C OWL Working Group, 'Owl 2 web ontology language: Document overview', Recommendation, W3C, (October 2009). Available at http://www.w3.org/TR/owl2-overview/.

[13] *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, eds., Jeff Z. Pan, Guido Vetere, José Manuél Gómez-Pérez, and Honghan Wu, Springer, 2017.

[14] Axel Polleres, 'From SPARQL to rules (and back)', in *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, eds., Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, pp. 787–796. ACM, (2007).

[15] Axel Polleres and Johannes Peter Wallner, 'On the relation between sparql1. 1 and answer set programming', *Journal of Applied Non-Classical Logics*, **23**(1-2), 159–212, (2013).

[16] Raymond Reiter, 'On closed world data bases', in *Logic and Data Bases*, eds., Hervé Gallaire and Jack Minker, 55–76, Springer, (1978).

[17] Gottfried Schenner, Stefan Bischof, Axel Polleres, and Simon Steyskal, 'Integrating distributed configurations with RDFS and SPARQL', in *Configuration Workshop*, volume 1220, pp. 9–15, (2014).

[18] Gottfried Schenner and Richard Taupe, 'Techniques for solving large-scale product configuration problems with ASP', in *Proceedings of the 19th International Configuration Workshop*, eds., Linda L. Zhang and Albert Haag, pp. 12–19, La Défense, France, (2017).

[19] Andy Seaborne and Steven Harris, 'SPARQL 1.1 Query Language', Recommendation, W3C, (March 2013). Available at https://www.w3.org/TR/sparql11-query/.

[20] Amit Singhal, 'Introducing the knowledge graph: things, not strings', *Official Google Blog*, (2012). Available at https://www.blog.google/products/search/introducing-knowledge-graph-things-not.

[21] Timo Soininen, Juha Tiihonen, Tomi Männistö, and Reijo Sulonen, 'Towards a general ontology of configuration', *AI EDAM*, **12**(4), 357–372, (1998).

# Measuring the Complexity of Product Configuration Systems

**Amartya Ghosh** and **Katrin Kristjansdottir** and **Lars Hvam**[1] and **Élise Vareilles**[2]

**Abstract.** The complexity of product configuration systems is an important indicator of both development and maintenance effort of the systems. Existing literature proposes a couple of effort estimation approaches for configurator projects. However, these approaches do not address the issues of comprehensibility and modifiability of a configuration model. Therefore, this article proposes a metric to measure the total cognitive complexity of the configuration model corresponding to a product configuration system, expressed in the form of an UML class diagram. This metric takes into account the number and the type of attributes, constraints and the relationships between classes in an UML class diagram. The proposed metric can be used to compare two configuration models, in terms of their cognitive complexity. Moreover, a relation between development time for a PCS project and the total cognitive complexity of the corresponding configuration model is established using linear regression. To validate the proposed approach a case study is conducted where the cognitive complexity is calculated for two configuration models.

## 1 Introduction

Information technology tools, such as product configuration systems (PCSs), are widely used to handle the increased amount of information shared amongst customers, sales and production departments at companies, arising out of an increase in the demand of product customization [1]. PCSs are knowledge-based IT systems which fulfil a configuration task. A configuration task is a special type of design activity [2] facilitated by a number of components, their corresponding properties and ports, and constraints which restrict the number of feasible combinations associated with the components [3].

PCSs contain detailed information about the companies' offerings. The information included in the configuration models depend on the type of the configuration system and usually include different components, attributes and rules of how the different components can be combined. Generally, the PCSs are not standalone IT systems. These systems are linked to existing IT systems within the companies (e.g., ERP, CAD, PLM, PDM, and calculation systems) either through integrations and/or interfaces. Coupled with the inherent complexity of the product knowledge modelled into the system, the integrations to other IT systems will render these configuration systems to be highly complex.

Practitioners in the industry are interested in metrics for comparing different PCS [5] and predicting the resource consumption required for developing, maintaining and extending the systems [6], [7]. The approach proposed in [6] defines a load estimation function that takes into account the impact of the industry and the organization on the project and the size and complexity of the product under consideration. The activities associated with the PCS development project include modelling the product structure and programming this model into a configuration software. However, this approach does not explicitly take into the consideration the complexity of the configuration model, while estimating the load required. Another approach to estimating the costs associated with PCS projects is to calculate the functional size of the UML class diagram of the configuration knowledge base using the IFPUG Function Point Analysis (FPA) technique [7]. This approach, however, is subjective in nature with regards to the definition of internal and external logical files (ILFs and ELFs). Moreover, the function point approach does not clarify the definition of one unit of function point [8]. The parametric complexity approach proposed in [5] involves calculating a metric on the basis of the number of business rules and the number of attributes, across two categories: field of engineering (sales, engineering and/or both) and the integration to other IT systems. However, the parametric complexity approach fails to take into account the complexity associated with the manner in which the configuration problem has been modelled into a PCS. A method to analyse the complexity of PCS is, therefore, needed.

This paper proposes a complexity metric that captures the different aspects associated with the PCS by focusing on the cognitive complexity of the configuration model. The metric can be used to effectively compare different PCS both developed on the same configuration software platforms or on different platforms. Also, the metric can be used to identify the impact of the complexity on the resource consumption for the development and maintenance activities associated with the PCS. Given that resources required for these activities are valuable and have limited availability, it is imperative that practitioners in the field have some metrics for predicting the extent of resource consumption required.

The proposed complexity metrics build on the approaches proposed in Felfernig (2004) and Kristjansdottir et al. (2017) by ascertaining the cognitive complexity of the UML class diagram associated with a given configuration problem. The proposed metric takes into account the effort required to understand and modify the way in which the PCS has been modelled, in terms of generalizations and aggregation structures, and the number and the type of business rules and attributes present in the configuration model.

The remainder of the paper is structured as follows. In Section 2 the concept of cognitive complexity known of software systems is introduced. The research methodology employed to carry out the analysis is explained in Section 3. Section 4 provides an overview of the mapping of the cognitive complexity metric of software

[1] Engineering Management Department, Technical University of Denmark, email: amgho@dtu.dk, katkr@dtu.dk, lahv@dtu.dk
[2] Toulouse University, email: elise.vareilles@mines-albi.fr

systems to configuration models. Section 5 then presents the findings from applying the method in a case company. Finally, Section 6 discusses the results and provide guidance for further studies.

## 2 Theoretical Background

This section presents the theoretical background for the proposed complexity metric for configuration models. It provides an overview of the existing literature on the topic of cognitive complexity of software systems.

Cognitive complexity is a measure of the functional complexity associated with designing and understanding a software system, based on the basic control structures (BCSs) existing in the system [9]. BCSs are considered to the building blocks of any software system. They are a collection of fundamental flow control mechanisms which are necessary for constructing the logical architecture of a software system [8]. Initially, Hoare et al. [10] identified three types of primitive commands (SKIP, ABORT and Assignment) and five types of more complex commands (sequential composition, non-determinism, conditional, iteration and recursion) in programming languages. Later, two further BCSs (function call, interrupt) were identified in system modelling by Wang [11]. Considering the sequential BCS as being representative of a unit BCS, a different cognitive weight is assigned to each BCS. The cognitive weight of a BCS is considered to be proportional to the effort required by a user in understanding the functionality and the semantics of the BCS, relative to the sequential BCS. Table 1 summarizes the BCSs and the corresponding cognitive weights:

**Table 1:** Cognitive weights of basic control structures (Adapted from Wang, 2006 [9]

| Category of BCS | BCS | Cognitive Weight |
|---|---|---|
| Sequential | Sequence (SEQ) | 1 |
| Branch | If-Then-Else (I-T-E) | 2 |
| | Case | 3 |
| Iteration | For-do | 3 |
| | Repeat-until | 3 |
| | While-do | 3 |
| Embedded | Function Call (FC) | 2 |
| | Recursion (REC) | 3 |
| Concurrency | Parallel (PAR) | 4 |
| | Interrupt (INT) | 4 |

Existing literature proposes several approaches for calculating the cognitive complexity of programs written in object-oriented programming languages, based on the cognitive weights of the BCSs. Object-oriented programs comprise a number of classes, each of which consist of a number of attributes and methods, and are linked to other classes through aggregation, generalization or association structures. The overall complexity of object-oriented programs is obtained by calculating the cognitive complexity of the constituent classes based on the relationships between the classes. The class complexity (CC) is calculated by taking into consideration the cognitive complexity of the individual methods, which in turn are made up of the blocks of BCSs, and the constituent attributes [12]. Therefore, the cognitive complexity of a method comprising q linear blocks of m layers of nesting BCSs with n linear BCSs in each layer, is calculated using the following equation (1):

$$MC = \sum_{i=1}^{q}[\prod_{k=1}^{m}\{\sum_{i=1}^{n} w_c(j,k,i)\}] \quad (1)$$

Where,
MC: total cognitive weight of the method
$w_c(j, k, i)$: cognitive weights of individual BCSs

Moreover, the attributes of a class are assigned cognitive weights according to a categorization of the cognitive phenomena associated with their datatypes [13]. The cognitive weights for the attribute categories are summarized in Table 2. The attribute complexity (AC) of a class is calculated by multiplying the number of attributes belonging to a particular datatype category and its corresponding cognitive weight.

**Table 2:** Cognitive weights assigned to attributes (Adapted from Arockiam and Aloysius [13])

| Attribute Datatype | Associated Cognitive Phenomena | Cognitive Weight |
|---|---|---|
| Primary | Sub-concious cognitive function | 1 |
| Derived | Meta cognitive function | 2 |
| User-defined | Higher cognitive function | 3 |

Therefore, for a class containing 'm' number of methods, with the i-th method having a complexity of MCi, and having an attribute complexity, AC, equation (2) shows the cognitive complexity of the class (CC).

$$CC = AC + \sum_{i=1}^{m}[MC_i] \quad (2)$$

The reusability of software code in object-oriented programming languages is facilitated through the concept of inheritance. However, the use of several levels of inheritance often has an adverse impact on the maintainability and comprehensibility of software systems [14], [15]. Therefore, a number of cognitive complexity metrics have been proposed to penalise the use of inheritance structures in OOPs [14], [16].

At the class level, the cognitive complexity of a class, $CC_i$, takes into account the number and complexity of the inherited classes, as shown in the following equation (3) [14][17]:

$$CC_i = \sum_{i}^{k} {}_{from} CC_{i\ from} + \sum_{i=1}^{l} MC_j \quad (3)$$

Where,
$CC_i$: cognitive complexity of the *i*-th class due to inheritance
$CC_{i\ from}$: cognitive complexity of an inherited class of class *i*
*k*: the number of inherited classes of class *i*
*l*: the number of constituent methods in class *i*
$MC_j$: the cognitive complexity of the *j*-th method of class *i*

Another approach to account for the presence of inheritance structures involves changing the way in which the total cognitive complexity of the program is calculated, depending on the level of inheritance of individual classes [16]. In this case, if the classes are on the same level of inheritance, their cognitive complexity

values are added together. However, if the classes are children of parent classes, then their values are multiplied. This is shown in the following equation (4) :

$$TC = \prod_{i=1}^{m} \lceil \sum_{k=1}^{n} WCC_{ik} \rceil \quad (4)$$

Where,
$TC$: cognitive complexity of the program
$WCC_{jk}$: cognitive complexity of k-th class at level j of inheritance.

The cognitive complexity approach has been applied to empirically calculate the complexity of UML class diagrams. The cognitive complexity of UML class diagrams has been shown to be correlated to the generalization and association structures, with an increase in the number of classes and attributes leading to an increase in the cognitive complexity and also the time required to understand and modify the class diagrams [18]. This implies that, given the UML class diagram of a configuration model and, consequently, the cognitive complexity of the configuration model, the required man-hours for developing and maintaining the PCS can be estimated. In turn, this will make it possible for better resource planning for PCS projects, allowing more accurate business cases to be developed, prior to the initiation of projects.

## 3 Methodology

The authors have adopted the case study research method to calculate the complexity of a configuration model and, subsequently, propose a relation, which expresses the development time in terms of the proposed complexity metric.

The case company is a world leader in catalysts and surface science. It offers a variety of catalysts and a complete range of proprietary equipment, spare parts, and consumables. The company first launched a PCS in 2013 and has since then been building up the configuration area at the company. Currently the company uses five PCSs in the company, while two configurators are under development. The PCSs used by the company support the sales process of both catalysts and equipment at the company and where the first PCS supporting the engineering process, or the detail design of an equipment is being tested. The complexity of the PCSs used at the company has quite variations but the lack of method for analysing the complexity makes it difficult both to predict maintenance effort for the different PCS in utilization at the company and development effort required for new PCS. Thus, the company's challenges were aligned with the research focus of the article.

In this study, the authors have analysed the configuration models pertaining to two PCSs. For each PCS, the following dataset was extracted from the configuration platform and used in the analysis:

- Model properties and statistics
  - Number of business rules
  - Number of attributes
  - Number of classes
- Documentation of the configuration model
  - Class structure
  - The attributes and their domains for each class
  - The business rules for each class

These parameters are utilized to calculate the cognitive complexity metric for each configuration model.

Moreover, the information regarding the development time and maintenance time for both the PCSs were also collected, in order to establish a relation between the cognitive complexity metric and the development time and maintenance time. However, the time registrations for these PCSs are not accurately maintained by the company, thereby requiring an assessment of the reliability of the data. After an initial analysis of the data, coupled with discussions with the members of the configuration team at the case company, the authors decided to analyse the two PCSs for which the time registrations were accurate and reliable. The relation between the development time for the PCSs, the cognitive complexity metrics and the unknown time constants were framed in the form of an objective function. The objective functions were solved to obtain the values for the time constants, by means of linear regression.

## 4 Proposed Cognitive Complexity Metric for Configuration Models

As addressed earlier, existing literature proposes a number of approaches for measuring the cognitive complexity of OOPs. However, the application of this concept to UML class diagrams of product configuration models presents an important deviation from those pertaining to OOPs. As PCSs are knowledge-based systems, the complexity of business rules (BRs) have a vital impact on the development and maintenance efforts of these systems [7]. In order to account for the impact of the BRs, the authors have assumed that each BR corresponds to a single method with a single block of layered BCS. Each constituent BCS of a BR has been assigned a cognitive weight, based on the classification proposed by Wang [8]. Table 3 summarises the cognitive weights for BCSs for each BR.

**Table 3:** Cognitive weight of BCS for BR

| Category of BCS | BCS | Cognitive Weight |
|---|---|---|
| Assignment | Assignment (ASS) | 1 |
| Branch | If-Then-Else (I-TE) or implication | 2 |
| Iteration | Iteration | 3 |
| Embedded | User Function Call (UFC) | 2 |
| | Standard Function Call (SFC) | 1 |

Thus, the cognitive complexity of a BR (BRC), consisting of m layers of nesting BCSs with $n_j$ linear BCSs in each layer, can be calculated using the following equation (5):

$$BRC = \prod_{i=1}^{m} \lceil \sum_{i=1}^{n_j} w_c(j,i) \rceil \quad (5)$$

Where,
$w_c(j, i)$: cognitive weight of the i-th BCS in the j-th layer of a nesting BCS pertaining to a BR.
$n_j$: number of BCS in the j-th layer of a nesting BCS pertaining to a BR

Another feature of UML class diagrams specific to configuration models is the cardinality of the classes representing the aggregation structure. This particular feature is accounted for

by the presence of the iteration BCS. A nested BR containing an iteration BCS would only be invoked in cases wherein more than a single instance of a particular class is required to be created.

As mentioned earlier, the cognitive complexity of a class also comprises the cognitive complexity of the attributes. Table 4 shows the possible attribute domains present in the configuration platform used at the case company categorised according to the datatype categories mentioned in Table 2.

**Table 4:** Cognitive Weights of Attribute Datatypes

| Datatype Category | Attribute Domain | Cognitive Weight |
|---|---|---|
| Primary | Integer, Float, Boolean | 1 |
| Derived | Named Domain | 2 |
| User-defined | Function, Class | 3 |

Therefore, given a configuration model, the cognitive complexity of a constituent class (CCi), comprising 'm' methods, 'n' attributes and 'p' business rules, is calculated by adding the cognitive complexities of the methods ($MC_j$), attributes (AC) and the business rules ($BRC_k$), as shown in equation (6):

$$CC_i = AC + \sum_{i=1}^{m} MC_i + \sum_{k=1}^{p} BRC_k \quad (6)$$

However, as the source codes of the configuration platforms (such as Tacton, Configit, SAP Configurator and so on) are generally confidential in nature and not available publicly, the authors decided to consider only the attribute complexity and the complexity of the business rules. Therefore, the final form of the cognitive complexity metric for a class used in the analysis is shown in equation (7).

$$CC_i = AC + \sum_{k=1}^{p} BRC_k \quad (7)$$

As an example to illustrate the various calculations, an excerpt of an UML class diagram, depicting the configuration model pertaining to a bicycle, is presented in Figure 1. The model comprises three classes (class Frame, class Suspension and class Fork). The model comprises two aggregation relationships (wherein class Suspension and class Fork are "parts of" the class Frame) but no inheritance structures. The corresponding cardinalities, attributes and business rules are also shown in the figure.

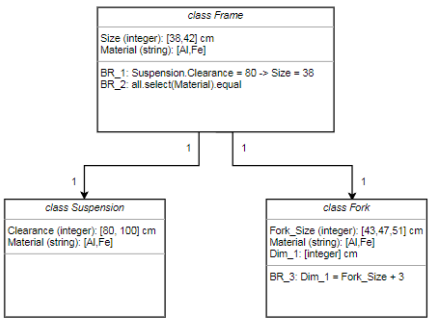


**Figure 1:** Descriptive example of bicycle

In case of the class Frame in Figure 1, the calculation for the attribute complexity is shown below in Table 5.

**Table 5:** Descriptive example of bicycle: Calculation of attribute complexity

| Attribute Name | Domain | Datatype category | Cognitive Weight | Attribute Complexity (AC) |
|---|---|---|---|---|
| Size | Integer | Primary | 1 | 1+2 = 3 |
| Material | Named Domain | Derived | 2 | |

For calculating the BRC for the class Frame, the nature of the BRs are established and the component BCSs are assigned their cognitive weights based on the categorization of BCSs.

BR_1 comprises of a single layer of nesting BCS (if statement) consisting of a single assignment BCS within it. Therefore, the BRCBR_1 is calculated using equation (5) as shown below:

*$BRC_{BR\_1}$ = (Cognitive weight of if statement) * (Cognitive weight of assignment statement)*
*$BRC_{BR\_1}$ = 2*1 = 2*

BR_2, on the other hand, is categorized as a standard function in the PCS software which has been used to model the configuration problem. Therefore, the business rule complexity of BR_2, BRCBR_2, is assigned the cognitive weight of 1. Therefore, the cognitive complexity of the class Frame, CCFrame, is calculated as shown below, according to the equation (7).

$CC_{Frame} = AC_{Frame} + (BRC_{BR\_1} + BRC_{BR\_2}) = 3+(2+1) = 6$

Table 6 summarizes the results for the remainder of the classes shown in Figure 1.

**Table 6:** Descriptive example of bicycle: Calculation of class complexity

| Class Name | BRC | AC | CC |
|---|---|---|---|
| Frame | 3 | 3 | 6 |
| Suspension | 0 | 3 | 3 |
| Fork | 1 | 4 | 5 |

Depending on the inheritance structure, the total complexity of the configuration model (TC) can be calculated by a combination of the cognitive complexities of the constituent classes, as shown in equation (8).

$$TC = \prod_{i=1}^{m} \lceil \sum_{k=1}^{n} CC_{ik} \rceil \quad (8)$$

Where,
$CC_{jk}$: cognitive complexity of the k-th class at the j-th level of inheritance.

Therefore, the total complexity of the configuration model shown in Figure 1 is given by: TC = 6+3+5 = 14. Given the value of the total cognitive complexity metric, TC, for a particular configuration model, the authors propose a relation, as shown by equation (9), to estimate the development time ($X_{Time}$) and the maintenance time for the model in man-hours.

$$X_{Time} = a + b.TC \quad (9)$$

Where,
$a$: development time constant (in man-hours)
$b$: development time constant (in man-hours) associated with a configuration model having a cognitive complexity of 1.

Taking into consideration the development times and the cognitive complexities of the two configuration models for which the time registration is available and accurate, the values of the time constants, a and b, are calculated, using linear regression.

However, as the value of TC is often not readily available before a configuration model has been fully developed, an estimated development time, $XET$, can be calculated for a particular configuration model if the number of constraints and the number of attributes, as shown in equation (10), or the number of classes, as shown in equation (11), are known.

$$X_{ET} = a + b.\ (\{\#_{attribute}.\ AC_{av} + \#_{BR}.\ BRC_{av}\}) \quad (10)$$

$$X_{ET} = a + b.\ (\#_{class}.\ CC_{avg}) \quad (11)$$

The following section presents the results obtained from the application of the proposed metrics on two configuration models at the case company.

## 5 Results from the Case Study

As explained earlier, the authors have only considered two PCSs, for which reliable and accurate time registration data was available, for their analysis. To calculate the cognitive complexity metric of the corresponding configuration models, the following data was extracted from the configuration platform used at the case company:

- The UML class diagrams
- The attributes and their datatypes for each class
- The business rules associated with each class
- Statistics pertaining to the overall configuration model:
  - Number of classes in the UML class diagram
  - Total number of attributes in the UML class diagram
  - Total number of constraints in the UML class diagram

Based on this data, the total complexity, TC, of the two PCSs were calculated. The results, along with the input data are summarised in Table 7.

**Table 7:** Complexity calculations and statistics for the two PCSs analysed

| Model | #Classes | #Attributes | #Rules | Parametric complexity | Cognitive complexity |
|---|---|---|---|---|---|
| PCS_1 | 82 | 1462 | 940 | 2402 | 9731 |
| PCS_2 | 27 | 334 | 262 | 596 | 1637 |

Table 5 shows considerable differences in both the parametric and the total cognitive complexities (TC) of the analysed PCSs. This result is aligned with the initial scoping of the research of selecting PCSs representing both high complexity values and low complexity values. Moreover, three additional metrics were also defined for each model, as shown in Table 8. These metrics include the average values of the attribute complexity ($AC_{avg}$), business rule complexity ($BRC_{avg}$) and the class complexity ($CC_{avg}$).

**Table 8:** Additional metrics representing the complexity of the PCS

| Model | ACavg | BRavg | CCavg |
|---|---|---|---|
| PCS_1 | 2.24 | 6.87 | 118.67 |
| PCS_2 | 1.71 | 4.08 | 60.63 |
| Overall average for any PCS | 1.97 | 5.48 | 89.65 |

The values for $AC_{avg}$ and $BR_{avg}$ for each of the models do not take into consideration the class structure for the configuration models. Based on these three metrics, the estimated average attribute complexity, average business rule complexity and average class complexity values for a configuration model developed at the case company are 1.97, 5.48 and 89.65 respectively. However, the results also show that there is a great variation of the complexity metrics calculated for the two systems, thus, the average complexity might not be provide us with the best possible estimates for the complexity of configuration models at the case company. Moving forward, more PCSs at the case company will have to be analysed to obtain results that are more indicative of the average PCS complexity at the company.

For the calculation of the time constants, a and b, pertaining to the relation between the cognitive complexity of the two configuration models and the development time for the corresponding PCSs, as expressed in Equation (9), the development times (in man-hours) are noted in Table 9.

**Table 9:** Development time in man-hours for the analysed PCS in the case company

| Model | Development Time (in man-hours) |
|---|---|
| PCS_1 | 2219 |
| PCS_2 | 380 |

Therefore, given the value of the total cognitive complexity of a configuration model, TC, the estimated development time of a PCS at the case company can be calculated as:

$$X_{Time} = 8.065 + 0.227TC \quad (12)$$

The $AC_{avg}$, $BRC_{avg}$ and $CC_{avg}$ are obtained from the data for the PCSs that have been analysed in this paper. Therefore, the final form of the equations (10) and (11) to estimate the development time of PCSs in man-hours at the case company, given the expected number of attributes and business rules or the expected number of classes, are presented in equations (13) and (14) respectively:

$$X_{ET} = 8.065 + 0.227(\{\#_{attribute} \times (1.97) + \#_{BR} \times (5.48)\}) \quad (13)$$

$$X_{ET} = 8.065 + 0.227(\{\#_{class} \times (89.65)\}) \quad (14)$$

Therefore, the following parameters of the configuration model are the pre-requisites for calculating an estimated development time for the PCS:

- The estimated number of classes, or
- The estimated number of business rules and attributes.

Moreover, based on the hours used for development the required hours for maintenance can be calculated. Based on interviews in the company it is estimated that 30% of the total development hours are required on a yearly base for the maintenance activities.

## 6 Discussions and Conclusions

This paper proposes a metric for evaluating the complexity of a PCS, by taking into the consideration the UML class diagram of the corresponding configuration model. The proposed cognitive complexity metric takes into the account the effort required to understand and modify the way in which the configuration problem has been modelled. Furthermore, the authors also investigate the relation between the effort in man-hours required for the development of a new PCS and the proposed complexity metric. The results obtained in the present article aim to contribute to the field of PCS complexity and the impact of PCS complexity on the effort estimation associated with the development and maintenance of PCS projects.

The proposed metric for calculating the cognitive complexity of a configuration model builds on research in the fields of the cognitive complexity of object-oriented software systems [8], [9], [12], [13], complexity of PCS [5]–[7] and the cognitive complexity of UML diagrams [18]. The presence of business rules and a lack of material on the source codes of commercial configuration platforms render the direct application of existing cognitive complexity metrics to configuration models unviable.

By having a more standardized way of analysing the complexity of a configuration model should enable both more accurate resource estimations and allow for a comparison of different configuration models. The authors adopted a structured approach to analyze the cognitive complexity of a configuration model and relating it to the development and maintenance efforts. First, an UML class diagram is analysed, which should give information regarding total complexity of a configuration model (based on number of classes, the number and type of attributes, number and type of business rules). Second, the average values of the attribute complexity ($AC_{avg}$), business rule complexity ($BRC_{avg}$) and the class complexity ($CC_{avg}$) are calculated. Third, linear regression is used to analyse man-hours for already developed systems to predict the future workload for new PCS.

The proposed method is validated in a case company where two different PCS were selected. The two PCS represent both PCSs with high and low parametric complexity. The proposed metric for analysing the cognitive complexity of the PCS confirmed the variation in the complexity. Further, by analysing the man-hours used for developing the systems, a function of total complexity and man-hours is used to predict the workload for a new PCS. The testing of the proposed method confirmed its usability. Further, testing is planned with in the case company where all the PCSs will be analysed in more detail, in order to verify the accuracy of the estimated development effort in man-hours when starting new PCS projects.

There are a few limitations to the research presented in this article. The proposed cognitive complexity metric primarily addresses only those configuration models which can be presented in form of an UML diagram. Further research will have to conducted in order to investigate the viability of applying cognitive complexity concepts to declaratively modeled products. Moreover, in case of declarative BRs, the authors have used their discretion to assign cognitive weights to such BRs. In certain cases, the procedural equivalent of the declarative BR is assigned the corresponding cognitive weight, whilst in others, the authors have considered the declarative BR to be either a standard function of the PCS or a user defined function.

This study focuses on the complexity of the configuration model. This means that the complexity resulting from integrations to other IT systems are not considered. Also, complexity for generating user interfaces and output documents are not taken into the account in the present study. Future work will, therefore, focus on including these parameters to better capture the overall complexity associated with a PCS and not only the complexity of the configuration model.

## REFERENCES


[1] C. Forza and F. Salvador, “Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems,” *Int. J. Prod. Econ.*, vol. 76, no. 1, pp. 87–98, 2002.

[2] S. Mittal and F. Frayman, “Towards a generic model of configuration tasks,” *Proc. Elev. Int. Jt. Conf. Artif. Intell.*, vol. 2, pp. 1395–1401, 1989.

[3] A. Felfernig, G. E. Friedrich, and D. Jannach, “UML as domain specific language for the construction of knowledge-based configuration systems,” *Int. J. Softw. Eng. Knowl. Eng.*, vol. 10, no. 4, pp. 449–469, 2000.

[4] K. R. Ladeby and J. L. Pedersen, “Applying Product Configuration Systems in Engineering Companies : Motivations and Barriers for Configuration Projects,” DTU, 2009.

[5] K. Kristjansdottir, S. Shafiee, L. Battistello, L. Hvam, and C. Forza, “Complexity of Configurators Relative to Integrations and Field of Application,” *19th Int. Config. Work.*, 2017.

[6] M. Aldanondo and G. Moynard, “Deployment of Configurator in Industry: Towards a Load Estimation,” *ECAI 2002 Work. Config.*, pp. 125–130, 2002.

[7] A. Felfernig, “Effort Estimation for Knowledge-based Configuration Systems.,” *SEKE*, pp. 148–154, 2004.

[8] Y. Wang, “Cognitive Complexity of Software and its Measurement,” *Proc. 5th IEEE Int. Conf. Cogn. Informatics*, vol. 1, pp. 226–235, 2006.

[9] J. Shao and Y. Wang, “A new measure of software complexity based on cognitive weights,” in *Canadian Conference on Electrical and Computer Engineering*, 2003, vol. 2, pp. 1333–1338.

[10] C. A. R. Hoare *et al.*, “Laws of programming,” *Commun. ACM*, vol. 30, no. 8, pp. 672–686, Aug. 1987.

[11] Y. Wang, “The Real-Time Process Algebra (RTPA),” *Ann. Softw. Eng.*, vol. 14, no. 1–4, pp. 235–274, 2002.

[12] S. Misra, M. Koyuncu, M. Crasso, C. Mateos, and A. Zunino, “A Suite of Cognitive Complexity Metrics,” in *International Conference on Computational Science and Its Applications -2012*, 2012, pp. 234–247.

[13] L. Arockiam and A. Aloysius, “Attribute Weighted Class Complexity : A New Metric for Measuring Cognitive Complexity of OO Systems,” *World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 5, no. 10, pp. 1151–1156, 2011.

[14] D. Mishra and A. Mishra, “Object-oriented inheritance metrics in the context of cognitive complexity,” *Fundam. Informaticae*, vol. 111, no. 1, pp. 91–117, 2011.

[15] R. Harrison, S. Counsell, and R. Nithi, “Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems,” *J. Syst. Softw.*, vol. 52, no. 2–3, pp. 173–179, 2000.

[16] S. Misra, M. Koyuncu, M. Crasso, C. Mateos, and A. Zunino, “A Suite of Cognitive Complexity Metrics,” *Int. Conf. Comput. Sci. Its Appl. - ICCSA 2012*, vol. 7336, pp. 234–247, 2012.

[17] H. Li, “Dynamic analysis of Object-Oriented software complexity,” *2012 2nd Int. Conf. Consum. Electron. Commun. Networks, CECNet 2012 - Proc.*, pp. 1791–1794, 2012.

[18] M. Esperanza Manso, J. A. Cruz-Lemus, M. Genero, and M. Piattini, “Empirical Validation of Measures for UML Class Diagrams: A Meta-Analysis Study,” *LNCS*, vol. 5421, pp. 303–313, 2009.

# Generating Configuration Models from Requirements to Assist in Product Management – Dependency Engine and its Performance Assessment

**Juha Tiihonen**[1] and **Iivo Raitahila**[1] and **Mikko Raatikainen**[1] and **Alexander Felfernig**[2] and **Tomi Männistö**[1]

**Abstract.** Requirements engineering is often, especially in the context of major open source software projects, performed with issue tracking systems such as Jira or Bugzilla. Issues include requirements expressed as bug reports, feature requests, etc. Such systems are at their best at managing individual requirements life-cycle. The management of dependencies between issues and holistic analysis of the whole product or a release plan is usually scantly supported. Feature modeling is an established way to represent dependencies between individual features, especially in the context of Software Product Lines — well-researched feature model analysis and configuration techniques exist. We developed a proof-of-concept dependency engine for holistically managing requirements. It is based on automatically mapping requirements and their dependencies into a variant of feature models, enabling utilization of existing research. The feature models are further mapped into a constraint satisfaction problem. The user can experiment with different configurations of requirements. The system maintains the consistency of dependencies and resource constraints. To evaluate the feasibility of the system, we measure the performance of the system both with some real and generated requirements. Despite some remaining performance issues, it seems that the approach can scale into managing the requirements of large software projects.

## 1 INTRODUCTION

There are various kinds of requirement management systems (RMS) applied in requirements engineering [10]. In particular, different issue tracker systems, in which requirements are captured as issues, are becoming increasingly popular, especially in large-scale, globally distributed open source projects, such as in the cases of Bugzilla for Linux Kernel, Github tracker for Homebrew, and Jira for Qt. An issue tracker can contain tens of thousands requirements, bugs and other items that are different ways interdependent from each other.

Issue tracker systems as RMSs provide primarily with support for individual requirements throughout various requirements engineering activities, such as requirements documentation, analysis, and management as well as tracking the status of a requirement over its life cycle. Even though individual dependencies, including more advanced constraints, can be expressed in the case of an individual requirement, more advanced analysis over all requirements of a system taking into account the dependencies and properties of the requirements is not well supported. For example, deciding a set of requirements to be implemented simultaneously might need to follow all dependencies transitively, which is not readily supported by the issue trackers. The issue trackers are not either necessarily optimal for the concerns of product or release management that need to deal with different requirement options, alternatives and constraints, as well as their dependency consequences when deciding what to do or not to do. However, dependencies in general are found to be one of the key concerns that need to be taken into account, e.g., in requirements prioritization [1, 9, 18] and release planning [2, 17]. In fact, the above concerns are not at the core of issue trackers' support for the requirements engineering activity. Rather, issue trackers focus more on a single issue, its properties, and its life cycle. The situation is not necessarily specific only for issue trackers, but it exists also in other kinds of RMS.

In the context of a Horizon 2020 project called OpenReq, we developed a proof-of-concept Dependency Engine for holistically managing requirements as a single model. It is based on automatically mapping requirements and their existing isolated dependencies into the Kumbang [3] variant of feature models, enabling utilization existing research. A feature model is further mapped into a constraint satisfaction problem. The user can experiment with different configurations of requirements. The system maintains the consistency of dependencies and resource constraints.

This paper outlines the principle of the Dependency Engine and addresses its feasibility in terms of performance. We measure the performance of the system both with some real and generated requirements. Responsive performance is important for interactive usage, e.g., what-if analysis of requirements to include in a release. Furthermore, it is important that decisions are based on current information; either relatively fast model generation or a way to update models 'on-the-fly' are required.

The rest of the paper is organized as follows. Section 2 outlines the concept of a feature model. Section 3 presents the research questions, general idea of the Dependency Engine, applied data and tests. Section 4 presents the results, Section 5 provides analysis and discussion. Finally, Section 6 concludes.

## 2 BACKGROUND: FEATURE MODELING

The notion of a *feature model*, similarly as a requirement, is not unambiguous. A *feature* of a feature model is defined, e.g., as a characteristic of a system that is visible to the end user [12], or a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among product variants [8]. A feature

[1] University of Helsinki, Finland, first.last@helsinki.fi
[2] Graz University of Technology, Austria, alexander.felfernig@ist.tugraz.at

model is a model of features typically organized in a *tree-structure*. One feature is the root feature and all other features are then the subfeatures of the root or another feature. Additional relationships are expressed by cross-branch *constraints* of different types, such as *requires* or *excludes*. Feature model dialects are not always precise about their semantics, such as whether the tree constitutes a part-of or an is-a structure [19]. Despite this, feature models have also been provided with various formalizations [8, 16] including a mapping to constraint programming [5, 6].

Specifically, we apply the Kumbang feature model conceptualization [3] as the basis. It has a textual feature modeling language and it has been provided with formal semantics. Kumbang specifies *subfeatures* as *part-of* relations and allows defining separate *is-a* hierarchies. Kumbang supports *feature attributes* and its *constraint language* can be used to express cross-branch relationships.

A feature model is a *variability model* roughly meaning that there are optional and alternative features to be selected, and attribute values to be set that are limited by predefined rules or constraints. When variability is resolved, i.e., a product is derived or configured, the result is a *configuration*. Variability is resolved by making configuration *selections* such as an optional feature is selected to be included, or one of alternatives is selected. A *consistent configuration* is a configuration in which a set of selections have been made, and none of the rules have been violated. A *complete configuration* is a consistent configuration in which all necessary selections are made.

Feature modeling has become a well-researched method to manage variability and has been provided with several different analyses to assist in system management [4].

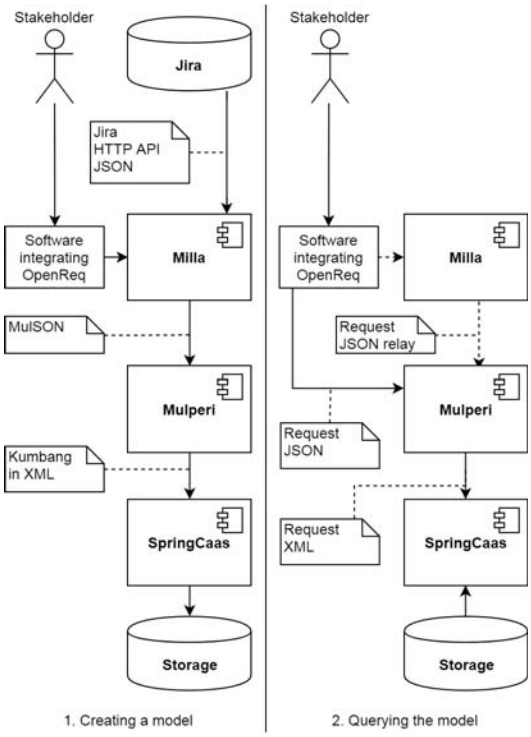


**Figure 1.** Workflows of the Dependency Engine

## 3 METHODS AND DATA

We follow Design Science in the sense that the aim is to innovate a novel intervention and bring it into a new environment so that the results have value in the environment in practice [11]. Dependency engine is the artifact of the intervention and this paper focuses on its quality attributes. The specific research questions are:

- **RQ1:** Can the OpenReq Dependency Engine scale to real-world projects?
- **RQ2:** How can the performance of the Dependency Engine be improved?

### 3.1 Approach and architecture

To facilitate requirement management via a feature-based approach, we make each requirement correspond to exactly one feature. The properties of a requirement correspond to the attributes of a feature. The dependencies of individual requirements are mapped to hierarchies and constraints of a feature model. We currently rely only on the dependencies explicitly expressed in requirements although we will aim to extract missing dependencies with NLP technologies. In order to make such a mapping, we need a feature model dialect that is conceptually relatively expressive supporting feature typing and attributes. Kumbang was selected for this purpose.

The Dependency Engine currently consists of three stand-alone software components with specific responsibilities: *Milla*, *Mulperi* and *SpringCaaS*, see Figure 1. There are two different workflows: creating a model from requirements data and making subsequently queries against the model. These three components operate as REST-type services and are implemented using the Java Spring framework[3].

Milla is a front-end that is used to access requirement data via volatile or case-dependent interfaces. For example, it extracts requirements via the API of Jira. It outputs MulSON, a JSON based intermediate transfer format understood by Mulperi.

Mulperi converts from a small number of stable requirement input formats such as MulSON into the Kumbang feature modeling language. It can generate a number of queries to SpringCaaS.

SpringCaaS takes as input Kumbang feature models and converts them into a corresponding Constraint Satisfaction Problem (CSP). Choco Open-Source Java library for Constraint Programming [15] was selected because it is Java-based, popular, and has good performance and a permissive license. The Kumbang model and corresponding the data structures are saved for subsequent use.

### 3.2 Potential bottlenecks and related tests

**Network and external system bottlenecks** Jira integration fetches requirements from the RMS one requirement at a time over network, which can potentially create performance bottlenecks. These bottlenecks are outside the scope of this paper[4].

**Requirement model generation** Milla generates feature models from requirements data fetched from Jira. Effectively, relevant data, such as IDs, dependencies and the attributes that are needed in inference, are extracted and a MulSON presentation is generated.

**Feature model generation** A requirement model expressed in MulSON is sent to Mulperi. Mulperi generates a feature model expressed in the Kumbang feature modeling language. Mulperi's func-

[3] https://spring.io/

[4] Bottlenecks were identified and solved by adding parallel connections.

tionality is largely based on data structure manipulation - JSON input and Kumbang output. The transformation is straightforward. Mulperi also saves the results into an in-memory database. This model is then sent to SpringCaaS in a single message.

**Feature model to CSP** A feature model expressed in Kumbang is parsed. Kumbang syntax resembles many programming languages. Therefore parsing is potentially heavy.

Based on the data structures representing the feature model, a corresponding Constraint Satisfaction Problem (CSP) is generated. Basically, a set of Boolean CSP variables represents instances of individual feature types. Each of these is related to corresponding integer CSP variables that represent attribute values of these individual features. Enumerated strings are mapped to integers. Choco constraints are created based on the dependencies; the constraints can access the presence of a feature, and relationships between attribute values of features. The current implementation supports only binary relationships (requires, excludes).

In addition, it is possible to specify global resource (sum) constraints over specific attributes. For example, the sum of efforts of included features can be constrained. To facilitate this, the implementation reserves attribute value 0 to attribute values of features that are NOT in configuration.

**CSP solving** The prime suspect for performance bottlenecks is solving the CSP representing a model of requirements. There are a number of tasks to accomplish based on a constructed model.

- check a configuration of features for consistency
- complete a configuration of features
- determine the consequences of feature selections

The selection of search strategy often has significant effect on solvability and quality of solutions [15].

## 3.3 Data

The performance evaluations are based both on real data from the Qt company and synthetic data.

### *3.3.1 Real requirements*

Qt is a software development kit that consists of a software framework and its supporting tools that are targeted especially for cross-platform mobile application, graphical user interface, and embedded application development. All requirements and bugs of Qt are managed in the Qt's Jira[5] that is publicly accessible. Jira[6] is a widely used issue tracker that can contain many issue types and has a lot of functionality for the management of issues. Issues and bugs can be considered as requirements and they have dependencies and attributes with constant values, such as priority and status. Thus, known requirements and their dependencies have already been identified and entered into Jira. Qt's Jira contains 18 different projects and although some of the projects are quite small and discontinued, QT-BUG as the largest project contains currently (April 2018) 66,709 issues.

For empirical evaluation with real data, a set of issues was gathered from Qt's Jira and processed through the whole pipeline. Only well-documented requirements having dependencies were selected to the dataset $JiraData$ that contains 282 requirements.

[5] https://bugreports.qt.io
[6] https://www.atlassian.com/software/jira

### *3.3.2 Synthetic data*

The synthetic datasets were created and run using automated scripts. $SynData1$ dataset contains a total of 450 models with permutations of the amounts of requirements (from 100 to 2000), a 'requires' dependency (from 0 to 75% of the requirements), an optional subfeature with one allowed feature (from 0 to 75% of the requirements) and a number of attributes (from 0 to 5 per requirement); each attribute has two possible values, e.g., 10 and 20.

A smaller dataset (60 test cases), $SynData2$, was used for optimization tests with sum constraints, see Section 3.4.5. $SynData2$ contains models with permutations of the amounts of requirements (from 100 to 2000), a 'requires' dependency (from 0 to 75% of the requirements), no further subfeatures and 1 or 2 attributes with a fixed random value from 1 to 100. An example of a $SynData2requirement$ in MULSON format:

```
{
"requirementId": "R4",
"relationships": [
{
"targetId": "R25",
"type": "requires"
}
],
"attributes": [
{
"name": "attribute1",
"values": ["9"],
"defaultValue": "9"
},{
"name": "attribute2",
"values": ["22"],
"defaultValue": "22"
}
],
"name": "Synthetic requirement nro 4"
}
```

## 3.4 Empirical tests

### *3.4.1 Test setup*

Measurements should be conducted when the software's behaviour is typical[13]. Since there is currently no production environment, the tests are conducted on a development environment that closely resembles the possible production environment. Furthermore, we aim to perform tests that could correspond to real usage scenarios.

The test machine is an Intel Core i5-4300U 1.9GHz dual core laptop with 16GB of RAM and an SSD disk, running Ubuntu Linux 16.04 and a 64-bit version of Oracle Java 1.8.0. All software components except for Jira are run on the same machine.

The examined software components log execution times to files that are collected after each automated test run. A timeout was set to limit the solving of Choco in SpringCaaS.

Although SpringCaaS is a single component, we often report the execution time in two parts: Choco Solver and the rest of SpringCaaS. This is because often Choco's solve operation takes the most time, but the initial tests showed that the Kumbang parser becomes a bottleneck in specific situations.

### *3.4.2 Initial trials and initial search strategy*

Initial testing was performed with the goal to complete a configuration of requirements with a minimal number of additional requirements. The pareto optimizer of Choco was applied to provide alternative solutions[7]. All features were included in the pareto front. By default, Choco uses the $domOverWDeg$ search strategy for interger and Boolean variables [15]. Table 3 describes the search strategies

[7] Pareto optimizer dynamically adds constraints: a solution must be strictly better than all previous solutions w.r.t. at least one objective variable [15].

applied. In our domain and way of modeling, the strategy effectively leads to selection of excessive number of features. This is contrary to the initial goal. As results in the beginning of Section 4 will show, an alternative search strategy was required to achieve satisfactory performance. The Search Strategy was changed to *minDomLBSearch*; it is used in the rest of the tests unless otherwise mentioned.

#### 3.4.3 Requirement model generation

$JiraData$ and $SynData1$ Datasets were applied to run the whole pipeline from gathered requirements to a Kumbang textual model and a serialized feature model. The process is illustrated at the left hand side of Figure 1. The different steps were timed.

In the case of $SynData1$ dataset, Milla was bypassed because the test cases were expressed directly in MULSON. Note that model generation includes finding a consistent configuration of features; this search is performed as a form of a model sanity check.

#### 3.4.4 Requirement configuration

Autocompletion of configurations was performed with the $JiraData$ dataset. A run was performed with a sum calculation constraint. Here, each requirement has a numerical priority attribute. The query instructed SpringCaaS to give a configuration where the sum of these priority attributes was greater than 100.

More substantially, requirement configuration was also performed with the $SynData1$ dataset to analyse the performance under varying number of requirements and their properties (attributes, dependencies), and user-specified requirements. This test applies optimization to find a (close to) minimum configuration that includes preselected features, if any. Effectively, the configuration of requirements is completed. This is presumably one of the computationally most intensive operations. The configuration phase is tested in ten iterations: first selecting only one requirement and then increasing the number of selected requirements by 10% so that the tenth iteration has 1 + 90% requirements selected.

#### 3.4.5 Optimised release configuration under resource constraint

We performed a number of resource-constrained optimization tests. Here, we applied global sum (resource) constraints specified in Table 1 to constrain the allowed solutions. $SynData2$ Dataset contains test cases with 1 or 2 attributes per requirement (see Section 3.3.2). Effectively, the combination of number of attributes and the applied constraint correspond to usage scenarios presented in Table 2. Finally, we applied the $bestBound(minDomLBSearch())$ search strategy, after we had experimented with different alternatives, see Section 3.4.6 and corresponding results.

We run the tests with 60s, 10s and 3s timeout values to see the effect of allowed time on the solvability and to get an impression on the quality of solutions. In addition, we developed and experimented with a custom algorithm that (roughly) first 'filled' effort bounds with 'big' features and used 'small' ones to meet the bound.

#### 3.4.6 Determining search strategy

We tested a set of different search strategies for performance, utilizing the 2000 requirement test cases of the $SynData2$ dataset. The experimented basic search strategies included $activityBasedSearch$, Choco default $domOverWDeg$, and $minDomLBSearch$, see Table 3. These were augmented with $bestBound$, $lastConflict$ or both; e.g., $bestBound$ adds directs search based on the objective function and a strict consistency check.

**Table 1.** Resource constraints for optimization tests

| Constraint# | Constraint |
|---|---|
| 0 | $\sum attribute1 > 1000$ |
| 1 | $\sum attribute1 = 1000$ |
| 2 | $\sum attribute1 < 1000$ |
| 3 | $\sum attribute1 > 1000 \wedge \sum attribute1 < 2000$ |
| 4 | $\sum attribute1 > 1000 \wedge \sum attribute2 < 2000$ |

**Table 2.** Constraints, number of attributes and usage scenarios

| Constraint# | #attributes | Optimization goal |
|---|---|---|
| 0, 1, 3 | 1 | Simulate achieving desired utility with a minimal number of requirements to implement. Minimizes the number of requirements. |
| 2 | 1 | Check the consistency of a given partial configuration with respect to maximum effort and complete the configuration with as few requirements as possible. Minimizes the number of requirements. In the case of $SynData2$, the test is redundant, only the root feature will be included. |
| 4 | 1 | (Not relevant, 2 attributes in constraint, 1 in model) |
| 0, 1, 2, 3 | 2 | Simulate maximisation of utility under resource constraint: Maximize sum of $attribute2$ |
| 4 | 2 | Minimize the number of requirements to implement under constraints of minimum utility and maximum effort. |

## 4 RESULTS

The results of the initial trials are in the two first rows of Table 4. The timeout and solution limits were disabled. The processing time was unacceptable, as reflected in the results.

### 4.1 Requirement model generation

The first two rows of Table 6 present the results of processing the $JiraData$ dataset through the whole pipeline. Table 5 shows the results of processing the $SynData1$ dataset. A save operation includes finding a consistent non-optimized configuration of requirements.

Figure 2 presents cases with 1000 requirements. Each bar color corresponds to a test case with a specific number of dependencies (from 0 to 200) and subfeatures (from 200 to 1000). The elapsed time in Mulperi, SpringCaaS and Choco are shown for 0, 2000 and 5000 attributes, that is, 0, 2 or 5 attributes per feature, each with two possible values per requirement.

Figure 3 depicts a case with 1000 requirements and different number of subfeatures (a requirement can be a subfeature of many requirements). Please note the logarithmic scale. With 5000 subfeatures it took over five hours to parse the model.

Starting from (some) models with 1000 requirements, the serialization of the parsed Kumbang model failed due to a stack overflow error. It was necessary to increase the Java Virtual Machines stack size to one gigabyte to prevent out-of-memory errors.

**Table 4.** Effect of search strategy with Pareto optimizer, $JiraData$ dataset

| Strategy | Optional features | Mandatory features | Attributes | Solutions | Time |
|---|---|---|---|---|---|
| default | 14 | 0 | 0 | 60 | 130 to 300 ms |
| default | 20 | 0 | 0 | 1046 | 11600 to 11900 ms (unacceptable) |
| minDomLBSearch | 14 | 0 | 0 | 1 | 120 to 170 ms |
| minDomLBSearch | 20 | 0 | 0 | 1 | 150 to 190 ms |
| minDomLBSearch | 235 | 0 | 2 per feature | 1 | 160 to 200 ms |
| minDomLBSearch | 118 | 117 | 2 per feature | 1 | 400 to 650 ms |

**Table 5.** Minimum, maximum and median test cases of the save phase, $SynData1$ dataset

| Requirements | Dependencies | Subfeatures | Attributes | Mulperi time (ms) | SpringCaaS time (ms) | Choco time (ms) | Total time (ms) |
|---|---|---|---|---|---|---|---|
| 500 | 375 | 200 | 0 | 85 | 158 | 98 | 341 |
| 500 | 50 | 500 | 1000 | 504 | 247 | 493 | 1244 |
| 500 | 250 | 500 | 2500 | 1971 | 439 | 2133 | 4543 |
| 750 | 563 | 150 | 0 | 159 | 220 | 401 | 780 |
| 750 | 375 | 0 | 1500 | 1040 | 371 | 2239 | 3650 |
| 750 | 563 | 1125 | 3750 | 4988 | 684 | 6359 | 12031 |
| 1000 | 750 | 400 | 0 | 309 | 347 | 670 | 1326 |
| 1000 | 750 | 0 | 2000 | 1895 | 584 | 4144 | 6623 |
| 1000 | 750 | 1500 | 5000 | 8859 | 1029 | 12772 | 22660 |
| 1500 | 1125 | 600 | 0 | 584 | 509 | 2009 | 3102 |
| 1500 | 0 | 1500 | 3000 | 4942 | 733 | 8756 | 14431 |
| 1500 | 750 | 2250 | 7500 | 21747 | 1738 | 30270 | 53755 |
| 2000 | 1000 | 800 | 0 | 661 | 566 | 4781 | 6008 |
| 2000 | 1500 | 0 | 4000 | 6958 | 1079 | 15816 | 23853 |
| 2000 | 1500 | 2000 | 10000 | 37692 | 2018 | 46433 | 86143 |

**Table 3.** Choco Search strategies

| Search strategy | Description |
|---|---|
| activityBasedSearch | Search strategy for "black-box" constraint solving. " ... the idea of using the activity of variables during propagation to guide the search. A variable activity is incremented every time the propagation step filters its domain and is aged."[14]. Used parameters (GAMMA=0.999d, DELTA=0.2d, ALPHA=8, RESTART=1.1d, FORCE_SAMPLING=1) [15] |
| domOverWDeg | Choco default. "Intuitively, it avoids some trashing by first instantiating variables involved in the constraints that have frequently participated in dead-end situations" [7]. Slightly oversimplifying, the strategy attempts to solve hard parts of a CSP first, weighting constraints by their participation in dead-ends. |
| minDomLBSearch | "Assigns the non-instantiated variable of smallest domain size to its lower bound" [15] |
| bestBound | Search heuristic combined with a constraint performing strong consistency on the next decision variable and branching on the value with the best objective bound (for optimization) and branches on the lower bound for SAT problems.[15] |
| lastConflict | "Use the last conflict heuristic as a plugin to improve a former search heuristic Should be set after specifying a search strategy."[15] |

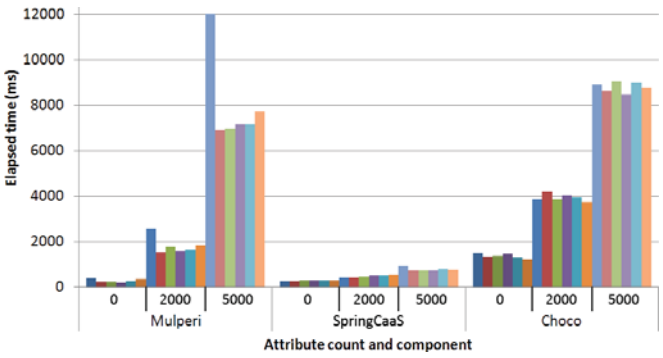


**Figure 2.** Performance effect of attributes

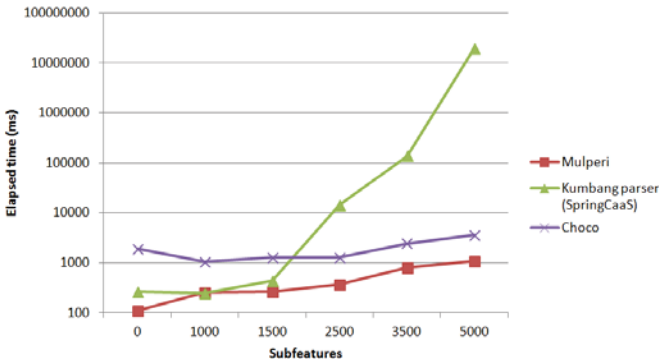


**Figure 3.** Kumbang parser's fatigue

**Table 6.** Measurements from the whole pipeline, $JiraData$ dataset

| Function | Requirements | Request | Milla time | Mulperi time | SpringCaaS time |
|---|---|---|---|---|---|
| Save model | 1 | - | 0,182 s | 0,010 s | 0,050 s |
| Save model | 282 | - | 1,252 s | 0,311 s | 0,315 s |
| Configure | 1 | empty | - | 0,050 s | 0,005 s |
| Configure | 282 | empty | - | 0,050 s | 0,143 s |
| Configure | 282 | 10 features | - | 0,040 s | 0,172 s |
| Configure | 282 | 25 features | - | 0,077 s | 0,127 s |
| Configure | 282 | 50 features | - | 0,116 s | 0,099 s |
| Configure | 282 | 10 features with dependencies | - | 0,040 s | 0,093 s |
| Configure | 282 | sum calculation constraint | - | - | 5,098 s (timeout) |

**Table 7.** Minimum, maximum and median test cases of the configuration phase, $SynData1$ dataset

| Requirements | Dependencies | Subfeatures | Attributes | Requirements in request | Mulperi (ms) | SpringCaaS (ms) | Choco (ms) | Total (ms) |
|---|---|---|---|---|---|---|---|---|
| 100 | 10 | 0 | 0 | 1 | 9 | 10 | 4 | 23 |
| 100 | 10 | 20 | 200 | 91 | 10 | 21 | 8 | 39 |
| 100 | 0 | 0 | 500 | 1 | 27 | 54 | 14 | 95 |
| 500 | 0 | 100 | 0 | 451 | 34 | 79 | 19 | 132 |
| 500 | 100 | 200 | 0 | 101 | 33 | 61 | 71 | 165 |
| 500 | 0 | 0 | 2500 | 201 | 34 | 266 | 549 | 849 |
| 750 | 0 | 150 | 0 | 601 | 60 | 122 | 55 | 237 |
| 750 | 0 | 150 | 1500 | 376 | 63 | 156 | 344 | 563 |
| 750 | 375 | 0 | 3750 | 601 | 70 | 273 | 1614 | 1957 |
| 1000 | 750 | 1000 | 0 | 1 | 129 | 133 | 126 | 388 |
| 1000 | 500 | 0 | 2000 | 401 | 90 | 252 | 777 | 1119 |
| 1000 | 0 | 0 | 0 | 1 | 1344 | 414 | 1788 | 3546 |
| 1500 | 0 | 0 | 0 | 1351 | 186 | 363 | 179 | 728 |
| 1500 | 0 | 0 | 3000 | 751 | 185 | 364 | 2159 | 2708 |
| 1500 | 300 | 0 | 7500 | 1351 | 263 | 715 | 9087 | 10065 |
| 2000 | 0 | 0 | 0 | 1801 | 237 | 619 | 334 | 1190 |
| 2000 | 200 | 0 | 4000 | 801 | 297 | 515 | 4445 | 5257 |
| 2000 | 1000 | 0 | 10000 | 1801 | 573 | 1056 | 16464 | 18093 |

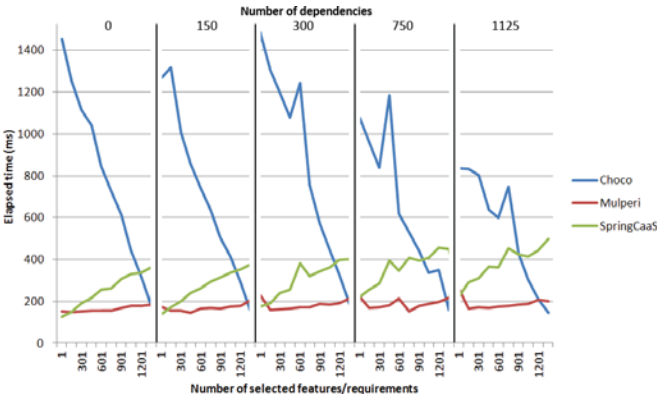


**Figure 4.** Performance effect of dependencies, 1500 requirements

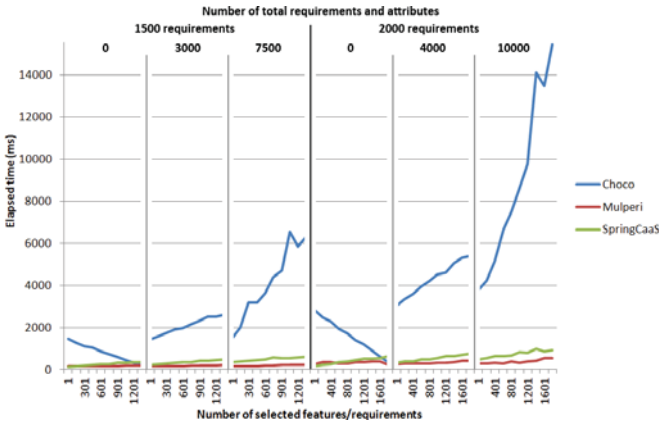


**Figure 5.** Performance effect of selected requirements and unselected attributes

## 4.2 Requirement configuration

The results of the configuration task with the $JiraData$ dataset are from row 3 onwards in Table 6. In case of the sum constraint (the last row), SpringCaaS was able to find 107 to 120 solutions before the timeout at 5s was reached.

Table 7 contains the minimum, maximum and median measurements of total execution times for varying numbers of requirements, dependencies, subfeatures, attributes and number of pre-selected requirements in the request.

Figure 4 shows the effect of the number of dependencies in case of 1500 requirements per test case, but with a varying number of requires-dependencies.

Figure 5 shows the effect of the number of requirements and attributes in case of 1500 and 2000 requirements per test case.

## 4.3 Optimized configuration under resource constraint

Table 8 presents a summary of the results of test that minimize the number of features. Note that constraint #4 is from test cases with 2 attributes, the others apply to test cases that have one attribute that is constrained. Because all features are optional, tests with constraint #1 trivially contain only the root feature of the model.

Table 9 represents the results of optimizing via Maximization of the sum of attribute 2 (e.g. utility) under constraints on $attribute1$.

Test cases with 100, 500, 750, 1000, 1500 and 2000 requirements and varying numbers of requirements are solvable with 60s timeout. 10s handles all cases except 2000 requirements. 3s timeout is only applicable to cases with 100, 500 and 750 requirements.

**Table 8.** Minimization of the number of features. Results of 60 second timeout compared with 10 and 3 second timeouts and the custom algorithm. Lower number of features in a solution is better. $Test$: the type of the testcases, $\#a$: the number of attributes in the test cases. $\overline{N}$: the average number of features in the minimal solution found with the 60s timeout. $N_{=_{10}}$: the number of test cases where 10s timeout search finds the same number of features than the 60s version. $N_{>_{10}}$: the number of test cases where 10s timeout search includes a larger number of features than the 60s version. $\overline{\Delta_{N_{10}}}$: the average number of additional features included in a solution found with 10s timeout when compared to the 60s search. $\overline{\Delta N_{10}}(\%)$: the average percentage of additional included features found by the 10s version. $N_{=_3}$, $N_{>_3}$, $\overline{\Delta_{N_3}}$, $\overline{\Delta N_3}(\%)$: 3 second timeout versions analogously as 10s. The corresponding figures of the custom algorithm are presented similarly: $N_{=_c}$, $N_{<_c}$, $N_{>_c}$, $\overline{\Delta_{N_c}}(\%)$. Note that $N_{<_c}$ is the number of cases where the custom algorithm finds a better solution. $SynData2$ dataset.

| $Test$ | $\#a$ | $\overline{N}$ | $N_{=_{10}}$ | $N_{>_{10}}$ | $\overline{\Delta_{N_{10}}}$ | $\overline{\Delta N_{10}}(\%)$ | $N_{=_3}$ | $N_{>_3}$ | $\overline{\Delta_{N_3}}$ | $\overline{\Delta N_3}(\%)$ | $N_{=_c}$ | $N_{<_c}$ | $N_{>_c}$ | $\overline{\Delta_{N_c}}(\%)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 14.3 | 14 | 11 | 0.48 | 3.38% | 7 | 8 | 0.60 | 4.92% | 4 | 7 | 19 | 2573.33% |
| 1 | 1 | 14.3 | 14 | 11 | 0.52 | 3.63% | 7 | 8 | 0.67 | 4.92% | 6 | 4 | 20 | 106.67% |
| 2 | 1 | 1.0 | 25 | 0 | 0.00 | 0.00% | 15 | 0 | 0.00 | 0.00% | 30 | 0 | 0 | 0.00% |
| 3 | 1 | 14.3 | 13 | 12 | 0.52 | 3.65% | 7 | 8 | 0.73 | 4.92% | 4 | 7 | 19 | 620.00% |
| 4 | 2 | 14.4 | 17 | 8 | 0.32 | 2.26% | 6 | 9 | 0.67 | 4.40% | 0 | 0 | 30 | 1456.67% |

A memory of 3 GB was required to complete the tests. The $bestBound$ search strategy became feasible by applying the optimization to one variable or to the sums of attributes. A pareto front with all feature variables caused excessive memory consumption.

## 4.4 Determining search strategy

Table 10 compares search strategies with 2000 requirement test cases and minimization tasks. $defaultSearch$ and $activityBasedSearch$ fail in a number of cases with 60s timeout.[8] $minDomLBSearch$ can solve all these cases. Negative $\Delta N$ indicates that the compared search strategy found better solutions (e.g. Total number of features was 18 less in the 30 tests) .

Constraint #2 with 2 attributes is essentially unconstrained for big problems. Here, the optimal solution includes all features. Plain $minDomLBSearch$ fails to 'notice' that. Both $bestBound(minDomLBSearch())$ and $lastConflict(bestBound(minDomLBSearch()))$ help the solver to find the maximal solution. Of these, in terms of maximized result on $attribute2$, $bestBound(minDomLBSearch())$ is slightly better in 3 cases and $lastConflict(bestBound(minDomLBSearch()))$ in one. Due to limitations of space, further details are omitted.

Earlier tests with all features in the pareto front prevented the usage of $bestBound$ strategy due to increased memory consumption.

**Table 10.** Comparison of search strategies with 2000 requirement cases and Minimization tasks with 60s timeout

| Search Strategy | # no solution | $\Delta N$ |
|---|---|---|
| $minDomLBSearch()$ | 0 | 0 |
| $lastConflict(minDomLBSearch())$ | 0 | -12 |
| $bestBound(minDomLBSearch())$ | 0 | -18 |
| $lastConflict(bestBound(minDomLBSearch()))$ | 0 | -18 |
| $defaultSearch()$ | 20 | |
| $bestBound(defaultSearch())$ | 45 | |
| $activityBasedSearch()$ | 50 | |
| $bestBound(activityBasedSearch())$ | 49 | |
| $lastConflict(bestBound(activityBasedSearch()))$ | 49 | |

# 5 ANALYSIS AND DISCUSSION

**Initial trials** The results of Table 4 turned out to be too good: it happens that the minimal requirement configurations of models in $JiraData$ are unique. That is, the solver can find a minimal solution with MinDomLBSearch and even prove its optimality.

[8] This test was performed with a different, weaker computer than the normally used one.

**Requirement model generation** The number of dependencies between the requirements seem to have no impact during the save phase. To avoid out-of memory errors, Kumbang model read and write methods could be overridden with an implementation that suits better for the Kumbang data structure, or the serialization phase could be omitted altogether. On the other hand, optimized solving needs even more memory.

Increasing the number of attributes increases the processing time of each component steadily, see Figure 2. Increasing the amount of subfeatures increases the processing time of Mulperi and Choco steadily as well, but when the amount of subfeatures is very large, the Kumbang parser slows down drastically, see Figure 3.

**Requirement configuration** The results in Section 3.4.4 suggest that a five second timeout would be sufficient for models with about 500 requirements or less. The configuration of all 1000 requirement models and most of the 1500 requirement models can be performed in less than five seconds.

The timeout value of the save phase could be set to be longer. Both timeout values could be controlled with parameters, for example if the user thinks that he/she can wait for a full minute for the processing to complete. During the configuration phase, the dependencies actually ease Choco's inference burden. Figure 4 with 1500 requirements shows that when there are no dependencies, the preselected requirements in the configuration request speed up Choco linearly.

The increase in configuration request size adds processing overhead to SpringCaaS. Secondly, when the dependency rate gets higher, more requirements are included in the configuration early on, again helping Choco perform faster. The same is true for subfeatures: selecting requirements with subfeatures decreases processing time.

With attributes, the situation is the opposite. The more there are attributes and the more configuration request contains selected requirements, the more time it takes to select attributes, see Figure 5.

The optimization task is computationally intensive. It is difficult for the solver to determine if an optimal solution has been found. Therefore solving practically always ends with a timeout.

**Optimised release configuration under resource constraint** When a solution is found, the versions with a lower timeout value remain almost as good as solutions obtained with 60s timeout. The custom algorithm was expected to perform well in test case types 1 and 2. However, this seems not be the case. Out of 150 test cases, the algorithm finds better solutions than the the 'normal' minimizer in 18 cases. In the clear majority of cases, it performs worse.

**Table 9.** Maximization of sum of attribute 2 (e.g. utility). Results of 60 second timeout compared with 10 and 3 second timeouts. The custom algorithm is excluded. Higher sum of attribute 2 ($a2$) is better. $Test$: the type of the testcases, $\#a$: the number of attributes in the test cases. $\overline{N_{60}}$: the average number of features in a solution found with the 60s timeout. $\overline{a1_{60}}$, $\overline{a2_{60}}$ :average value of attribute 1 / attribute 2 in solutions identified with 60s timeout, respectively. $N_{10,a2,<}$ and $N_{10,a2,=}$: the number of test cases where 10s timeout search finds a lower / same same sum of attribute 2 than the 60s version, respectively. $\overline{\Delta N_{10}(\%)}$ : the average difference (percentage) between number of included features between 60s and 10s timeout versions. $\overline{\Delta a2_{10}(\%)}$: the average difference (percentage) between sum of attribute 2 of included features between 60s and 10s timeout versions. 3 second timeouts are analogous, $SynData2$.

| $Test$ | $\#a$ | $\overline{N_{60}}$ | $\overline{a1_{60}}$ | $\overline{a2_{60}}$ | $N_{10,a2,<}$ | $N_{10,a2,=}$ | $\overline{\Delta N_{10}(\%)}$ | $\overline{\Delta a2_{10}(\%)}$ | $N_{3,a2,<}$ | $N_{3,a2,=}$ | $\overline{\Delta N_3(\%)}$ | $\overline{\Delta a2_3(\%)}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 976 | 48979 | 49255 | 0 | 25 | 0.0% | 0.0% | 0 | 15 | 0.0% | 0.0% |
| 1 | 2 | 33.2 | 1000 | 1821 | 24 | 1 | -2.1% | -4.1% | 15 | 0 | -2.9% | -5.9% |
| 2 | 2 | 33.5 | 998 | 1831 | 21 | 4 | -3.4% | -4.6% | 13 | 2 | -5.7% | -6.5% |
| 3 | 2 | 53.6 | 1998 | 2860 | 23 | 2 | -2.1% | -3.2% | 15 | 0 | -3.6% | -4.5% |

**Determining search strategy** The best search strategy for our purposes is $bestBound(minDomLBSearch())$ instead of plain $minDomLBSearch()$, because it provides slighly better results in minimization tests and maximizes significantly better.

## 6 CONCLUSIONS

Solutions without optimization are easy to find; solvers such as Choco have an easy task with sparse dependencies. Still, at least for optimization, the selection of a search strategy matching the problem at hand remains crucial. It was surprising that the "black-box" activityBasedSearch[14] and Choco default domOverWDeg[7] did not perform in a satisfactory way.

The prototype engine easily scales to around 2000 requirements, even when optimization is desired. Despite some remaining performance issues, it seems that the approach can scale into managing the requirements of large software projects, even for interactive use.

However, very large software projects, such as QT-BUG remain challenging. A more close examination of the Qt Jira is required, because it seems that performance can be managed in various ways. First, there are different types of issues such as bugs and requirements that do not need to be considered at the same time. Second, Qt has used Jira over a decade and there is a lot of historical data. The rate of new Jira issues seems to be up to 20 per a day. So, considering only issues created or modified within three years would significantly decrease the amount of data. Third, the exact nature of Qt data and practical applications need to be inspected in more detail; now it seems that only about 10% of issues have dependencies, and the compositional hierarchy such as epics decomposed to smaller items needs a few levels at most.

The concept of Dependency Engine is novel and it seems to be feasible for its intended use for providing holistic support for the management of dependencies, also in the context of large software projects.

## ACKNOWLEDGEMENTS

This work has been funded by EU Horizon 2020 ICT-10-2016 grant No 732463. We thank the Qt Company for sharing the data.

## REFERENCES

[1] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Nazri Mahrin, 'A systematic literature review of software requirements prioritization research', *Information and Software Technology*, **56**(6), 568–585, (2014).

[2] David Ameller, Carles Farré, Xavier Franch, and Guillem Rufian, 'A survey on software release planning models', in *Product-Focused Software Process Improvement*, (2016).

[3] Timo Asikainen, Tomi Männistö, and Timo Soininen, 'Kumbang: A domain ontology for modelling variability in software product families', *Advanced Engineering Informatics Journal*, **21**(1), (2007).

[4] D. Benavides, S. Segura, and A. Ruiz-Cortes, 'Automated analysis of feature models 20 years later: A literature review', *Information Systems*, **35**(6), 615–636, (2010).

[5] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortes., 'Automated reasoning on feature models', in *17th Conference on Advanced Information Systems Engineering (CAiSE)*, (2005).

[6] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortes, 'Using constraint programming to reason on feature models', in *International Conference on Software Engineering and Knowledge Engineering*, (2005).

[7] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais, 'Boosting systematic search by weighting constraints', in *Proceedings of the 16th European Conference on Artificial Intelligence*, pp. 146–150. IOS Press, (2004).

[8] K. Czarnecki, S. Helsen, and U. W. Eisenecker, 'Formalizing cardinality-based feature models and their specialization', *Software Process: Improvement and Practice*, **10**(1), 7–29, (2005).

[9] Maya Daneva and Andrea Herrmann, 'Requirements prioritization based on benefit and cost prediction: A method classification framework', in *34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 240–247, (2008).

[10] Juan M. Carrillo de Gea, Joaqun Nicols, Jos L. Fernndez Alemn, Ambrosio Toval, Christof Ebert, and Aurora Vizcano, 'Requirements engineering tools: Capabilities, survey and assessment', *Information and Software Technology*, **54**(10), 1142 – 1157, (2012).

[11] S. Gregor, 'The nature of theory in information systems', *MIS Quarterly*, **30**(3), 611–642, (2006).

[12] K.C. Kang, S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson, 'Feature-oriented domain analysis (FODA) feasibility study', Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, (1990).

[13] D. Maplesden, E. Tempero, J. Hosking, and J. C. Grundy, 'Performance analysis for object-oriented software: A systematic mapping', *IEEE Transactions on Software Engineering*, **41**(7), 691–710, (July 2015).

[14] Laurent Michel and Pascal Van Hentenryck, 'Activity-based search for black-box constraint programming solvers', in *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pp. 228–243. Springer, (2012).

[15] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca, *Choco Documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. www.choco-solver.org, 2016.

[16] P-Y. Schobbens, P. Heymans, J-C. Trigaux, and Y. Bontemps, 'Generic semantics of feature diagrams', *Compututer Networks*, **51**(2), (2007).

[17] Mikael Svahnberg, Tony Gorschek, Robert Feldt, Richard Torkar, Saad Bin Saleem, and Muhammad Usman Shafique, 'A systematic review on strategic release planning models', *Information and Software Technology*, **52**(3), 237 – 248, (2010).

[18] R. Thakurta, 'Understanding requirement prioritization artifacts: a systematic mapping study', *Requirements Engineering*, **22**(4), 491–526, (2017).

[19] Juha Tiihonen, Mikko Raatikainen, Varvana Myllärniemi, and Tomi Männistö, 'Carrying ideas from knowledge-based configuration to software product lines', in *International Conference on Software Reuse*, pp. 55–62, (2016).

# Quasi-Finite Domains: Dealing with the Infinite in Mass Customization

**Albert Haag**[1]

**Abstract.** In this paper we propose to relax finiteness in relational tables and tabular constraints in a controlled way. We preserve the syntactic representation of a row in a table as a tuple of symbols. Some of these symbols refer to an atomic value as usual. Others, which we call *quasi-finite symbols* (QF-symbols), refer to infinite subsets of an underlying infinite domain. Practical examples for QF-symbols are references to (uncountable) real-valued intervals and *wildcards* representing countably infinite sets. Our goal is to provide a simple and smooth extension of the tabular paradigm, predominant in business, that is compatible with compression of the table to *c-tuples* [14] or to a *variant decomposition diagram* [11], and is amenable to constraint processing, such as local propagation.

The approach is based on organizing the QF-symbols pertaining to each product property in a *specialization relation* [8, 9]. A specialization relation is a partial ordering that expresses specificity of meaning. A QF-symbol can be ignored in the presence of a more special one.

To ensure that the sets represented by two distinct QF-symbols pertaining to the same domain are disjoint, we further require that it must be possible to represent the *intersection* and *set-differences* of QF-symbols. In order to be able to remove duplicates implicated by a disjunction of QF-symbols from result sets of queries, we require that it is possible to represent their *normalized set-union*.

QF-symbols may refer to any objects as long as the above requirements are met, e.g. regular expressions (unary predicates), rectangles (geometric shapes), etc.

## 1 Introduction

This work expands on a common theme: that data in tabular form is a natural, non-proprietary medium for communicating between interrelated business processes within an enterprise, as well as between enterprises. We focus on *mass customization* (MC), which we take to be *mass production* with a lot size of one. A non-configurable product, amenable to mass production, can have *product variants*[2]. For example, mass produced ballpoint pens come in several colors, but are otherwise identical. The offered colors are in a one-to-one correspondence with the manufactured ballpoint variants. The business attributes are maintained once for the generic ballpoint pen. Only one generic bill of materials that covers all possibilities needs to be maintained. For each variant the value of an additional product property, *color*, is needed to determine which ink filling and matching cap is used in assembling the variant[3].

MC adds *customization* to this setting by placing the emphasis on *individualization*, i.e. there will be many variants of a product and the business is prepared to produce only a single unit of each one on demand (lot size one). Accordingly, the product properties that distinguish the variants are central and potentially numerous[4]. In other work [13] we discuss the MC setting in more detail and show that compression of *variant tables*, tables listing combinations of product features, is a key element in managing the exponential explosion of the number of variants caused by the increase in the number of customization choices, which production technology now enables. Here, we propose to add to the expressivity of variant tables by presenting a *quasi-finite* (QF) framework that allows dealing with infinite sets of choices within the tabular paradigm.

If the domains of all descriptive properties are finite, then the number of variants is finite as well. Leaving the reference to the underlying generic MC product aside, each variant is defined by a value assignment to the properties, which we represent as a *relational tuple* (*r-tuple*). If the number of offered variants is not large, these r-tuples can be maintained as rows in a database table or spreadsheet, which then acts as a product model comprised of a single tabular constraint. If desired or needed, this overall *variant table* can conceptually be split into smaller tables that together form a *constraint satisfaction problem* (CSP). Each CSP variable corresponds to a product property and each CSP solution to an offered product variant[5]. We refer to any tabular constraint on product properties as a *variant table*.

Variant tables are a form of modeling that is very acceptable to a business. Their downside is that they may not scale with a growing number of choices for individualization. However, we show in [13] that expected regularities in the product variants will allow a compressed form of the table to scale. Here, our choice of the compressed form is a *variant decomposition diagram* (VDD) [10, 11], and the associated *c-tuples*, a term adopted from [14] and used there for a Cartesian product of sets of values.

However, infinite domains occur in MC practice, and neither tables of r-tuples nor classic CSP approaches allow infinite sets. In this paper we propose to relax finiteness in variant tables, and by extension in the associated constraint processing, in a controlled way.

[1] Product Management GmbH, Germany, email: albert@product-management-haag.de

[2] We use SAP terminology pertaining to the handling of products and product variants, citing [6] as a general reference. A brief sketch of the history of the SAP Variant Configurator is given in [7]

[3] The SAP tables relevant for configuration are listed in [6], Appendix A

[4] For simplicity of exposition, we disregard the possibility of needing to deal with variant structures, i.e. variants that have variants as parts. Our approach here addresses tabular data in general and could be extended to variant structures if needed.

[5] In practice, product models are not limited to use only tabular constraints. However, the reasoning here shows that product variants could be exclusively expressed in tables in the finite case. The single overall table listing all variants can be seen the the result of *compiling* the product model, e.g. to a *decision diagram* [2].

Our goal is to provide a simple and smooth extension of the tabular paradigm that retains its acceptance in business, and, particularly, allows compression to a VDD and c-tuples. We preserve the syntactic representation of a row in a table as a tuple of symbols, while allowing some of these, which we call *quasi-finite symbols* (QF-symbols), to refer to infinite sets. A symbol for a *real-valued interval*, which is uncountably infinite by definition, is an example of a QF-symbol. A *wildcard* symbol that refers to an infinite domain is also a QF-symbol. In contrast, a *wildcard* for a finite domain is just an *alias* for the finite set of values. We treat this as "syntactic sugar" and equivalent to the expanded set of values.

The approach we take here is illustrated by example in Section 4 and based on the following ideas[6]:

- Each property domain is defined by a finite set of symbols:
  - a finite domain by its *values*, which we refer to as *r-symbols*,
  - an infinite domain by one or more (disjoint) QF-symbols.
- We represent a value assignment to the product properties as a tuple of symbols. If the tuple contains only r-symbols, it is an r-tuple. If it also contains QF-symbols, we call it a QF-tuple. Both r-tuples and QF-tuples are interpreted a special cases of a c-tuple, where an r-symbol in the tuple is treated as a singleton set.
- We adapt the concept of a *specialization relation* from [8, 9] to QF-symbols. When queries or constraint solving need to consider two different QF-symbols for the same property simultaneously, they can ignore both symbols and focus instead on the more *special* symbol for their set intersection.
- Only a finite number of QF-symbols is needed, which can be derived in advance from the product model (e.g. the property domains and the variant tables)[7].
- Compression to a VDD, and through that to c-tuples, can be done as in the finite case, if we can ensure certain requirements are met.

One difference between a QF-symbol and an r-symbol is that the former still allows choice, i.e. it can be *specialized* or restricted further when required. The consequence is that some constraints may need to be formulated in non-tabular form, e.g. to express that for two real-valued properties *length* and *width* it should hold that: $length \geq width$. These constraints can be seen as inter-property predicates. Whereas we discuss unary intra-property predicates as QF-symbols, we will not deal with other inter-property predicates in this paper, except to note in passing that restricting real-valued intervals with numeric linear (in)equalities is an established technique (see Section 8.2) that can be smoothly integrated with our intended processing.

We show how configuration queries over variant tables with QF-symbols can be meaningfully supported. We also believe that the concept of specialization relations is an important bridge to constraint processing in general. The idea of defining a specialization relation via the *subset-relation* can be inverted: given a set of symbols from the column of a table that correspond to elements of a partial order, such that a unique greatest successor and a unique least common predecessor exists for any two elements, these symbols can be treated in a like manner to QF-symbols for purposes of queries and constraint processing, if we are willing to interpret the partial order as a specialization relation.

[6] This extends the simple processing of real-valued intervals and wildcards proposed in [10, 11] for a *set-labeled* VDD.

[7] If further QF-symbols are generated dynamically externally, the specialization relation will have to be extended dynamically. Nevertheless, at any given time a finite number of symbols will be needed.

As stated, the goal of this work is to smoothly extend the tabular paradigm, not to compete with other dedicated problem solving approaches, and we do not make any such comparisons here. The *quasi-finite* (QF) approach has not yet been tried in the field. Therefore, we cannot present results. Given that the VDD processing remains syntactically alike to the finite case, and given our positive experiences with specialization relations in other endeavors, we are confident that performance is not the issue. Instead, it will be a primary concern to establish usefulness in practice and evaluate acceptance by the business community.

The paper is structured as follows:

- We summarize a database approach to configuration in Section 2 and the topic of compression to VDDs and c-tuples in Section 3.
- We illustrate all ideas using an extensive example based on an MC T-shirt in Section 4.
- Constructing VDDs from QF-tuples is akin to constructing them from c-tuples. This topic is beyond the scope of this paper. However, we summarize the basic problem of ensuring disjoint c-tuples (QF-tuples) in Section 5.
- We look at the motivating examples of QF-symbols and how they meet our requirements in Section 6 in some detail.
- We discuss queries to variant tables with QF-symbols in Section 7.
- We present our ideas on specialization relations and their relation to constraint processing in Section 8. In particular we show that local propagation works seamlessly.
- We also believe that using QF-symbols (and perhaps c-tuples in general) directly in the definition of a product variant has business benefits, which we discuss in Section 9.
- We provide a summary and an outlook in Section 10.

## 2 Configuration in the Database Paradigm

The easiest MC business setting is when the business offering is a small finite set of product variants actually represented in extensional form in a relational database table or spreadsheet. Even when this is not possible, due to the size such a table would have, tabular constraints can be used to define the valid variants.

The extensional form of a tabular constraint naturally supports various data queries such as (1) and (2), here formulated in SQL, which are the most relevant for configuration as discussed in [11][8].

The query in (1) returns a result set of all variants matching the user's criteria[9]. The $k$ product properties are denoted by $v_1, \ldots, v_k$. The variant table is denoted as $\langle vtab \rangle$. $\langle R_j \rangle$ denotes a subset of the domain $D_j$ for product property $v_j$. The values of interest to a user when configuring can be communicated in the *WHERE* clause.

```
SELECT * FROM ⟨vtab⟩
       WHERE ⟨v1⟩ IN ⟨R1⟩ AND ...⟨vk⟩ IN ⟨Rk⟩;  (1)
```

The query in (2) returns the domain restriction for property $v_j$ under the *WHERE* clause.

```
SELECT DISTINCT ⟨vj⟩ FROM ⟨vtab⟩
       WHERE ⟨v1⟩ IN ⟨R1⟩ AND ...⟨vk⟩ IN ⟨Rk⟩;  (2)
```

[8] While the approach here may be extended to cover further SQL queries, this is beyond the scope of this paper.

[9] In the SQL syntax, an *IN* term in the *WHERE* clause need not be specified where no restriction is intended. However, for purposes of representing a query condition as a c-tuple (see Section 3), we will substitute $R_j = D_j$ for an omitted *IN* term

These queries can also be done to further filter the result sets of previous queries (see [11, 10]).

To sum up: tabular constraints in extensional form can be evaluated using database queries. In [11] we have shown that this extends to tables represented as VDDs in a way that also guarantees the efficiency of the queries. We now have to show here how to handle the queries (1) and particularly (2) in conjunction with QF-symbols.

## 3 C-Tuples, Table Compression, and Decision Diagrams

The discussion of compression in this section is illustrated with examples using a simple T-shirt in Section 4.

In the finite case a variant can be represented as an *r-tuple*. If we substitute sets for values in this tuple, the tuple is no longer relational, but represents the Cartesian set of all r-tuples that can be formed as combinations using values from the sets. We call such a Cartesian tuple a *c-tuple*[10]. As a tuple we denote it by $\mathbf{C} = \langle C_1, C_2, \ldots, C_k \rangle$, where $C_j \subset D_j$ and $D_j$ is the domain of the product property $v_j \in \{v_1, \ldots, v_k\}$. As a Cartesian set it would be written as $\mathbf{C} = C_1 \times C_2 \times \ldots \times C_k$.

In the context of the above definition, we don't care whether an element $C_j$ of a c-tuple is finite or infinite. Note that the set of r-tuples represented by a c-tuple is uncountable if one of the sets in the c-tuple refers to a real-valued interval.

The *WHERE* clause with the $k$ *IN* operators in (1) and (2) itself describes a c-tuple $\langle R_1, \ldots, R_k \rangle$, which expresses the set of values the user (the problem solving agent) believes in. We will refer to this c-tuple as the *query condition*. We will allow a query condition to be any c-tuple from our variant domain.

C-tuples offer a way to compress tables. For example, if the set of all variants is totally unconstrained, this can be represented by a single c-tuple, which is the Cartesian product of the product domains. With constraints, there will be more c-tuples, but often a c-tuple representation is much more compact than the extensional form [12]. For this reason, c-tuples are already used both formally and informally in configuration practice.

In the case of finite domains, a set of c-tuples can be further compressed to a decision diagram (DD). We use the form of a *Variant Decomposition Diagram* (VDD). As introduced in [10, 11], a VDD is a binary rooted *Directed Acyclic Graph* (DAG), where each node has a label denoting the assignment of a property to a value (r-symbol). Here we will allow QF-symbols in node labels as well. Each node has two emanating links, *HI* and *LO*, which we characterize as follows given a fixed ordering of the product properties: $v_1, \ldots, v_k$:[11]

- the *HI*-link of a node points to a node for the next product property $v_{j+1}$ or to the terminal sink $\top$ (*true*) from last column nodes.
- the *LO*-link points to an alternate value assignment for the same product property $v_j$ or to the terminal sink $\bot$ (*false*).

We will call a chain of nodes linked via *LO*-links an *l-chain*. If more than one QF-symbol appears in an l-chain, the QF-symbols must denote disjoint sets, in order to allow a unique decision for a node, given a value assignment.

Nodes in an l-chain that all have a common HI-link represent the disjunction of their value assignments and could be merged into one *set-labeled node*. In [10, 11] we introduced a VDD with *set-labeled* nodes, where a node was labeled with a finite set of r-symbols representing a disjunction of value assignments. Since any node labeled with such a finite set can be re-expanded into an l-chain of regular VDD nodes nodes that assign the symbols one at a time, we do not propose to use VDDs with set-labeled nodes in practice. We use them here in Section 4 to simplify the exposition.

A VDD is functionally equivalent to the extensional form of the table it represents from the perspective of the queries (1) and (2) relevant for configuration, see [11]. The extension of these queries to include QF-symbols is the topic of Section 7. A VDD can also support counting the number of tuples in a table or a result set of a query and access a tuple directly by its position in the table/result set.

## 4 T-Shirt Example

### 4.1 Classic Finite T-Shirt Variants

In [10] the concepts of representing a variant table using a VDD are illustrated using an example of a simple T-Shirt. We use this example here both to illustrate the concepts discussed so far, and also to illustrate the proposed extension to infinite sets.

The simple T-shirt has the three properties *Imprint* ($v_1$), *Size* ($v_2$), and *Color* ($v_3$) with the finite domains:

- $\{MIB$(Men in Black), $STW$(Save the Whales)$\}$
- $\{L$(Large), $M$(Medium), $S$(Small)$\}$
- $\{Black, Blue, Red, White\}$

Only 11 variants are valid due to constraints that state that $MIB$ implies $Black$ and $STW$ implies $\neg S$(Small). Table 1 is the extensional form of the variant table, which is small enough to be used as the only and definitive representation of the variants for the purposes of both business and configuration. It encodes the underlying CSP as a single tabular constraint.

The query (1) can be used to filter the variants to the set matching any given selection criteria (*query condition*) $\langle R_!, \ldots, R_k \rangle$. For example, if the user needs a small (*S*) sized T-shirt, there is only one solution (the first row in Table 1). Alternatively, if a *Red* T-shirt is desired, there are two variants that satisfy this (eighth and ninth rows), and the domains are restricted as follows: $Imprint \in \{STW\}$, $Size \in \{Medium, Large\}$, and Color $\in \{Red\}$ by applying the query (2) for each property in turn.

**Table 1.** Simple T-shirt

| Imprint | Size | Color |
|---|---|---|
| MIB | S | Black |
| MIB | M | Black |
| MIB | L | Black |
| STW | M | Black |
| STW | L | Black |
| STW | M | White |
| STW | L | White |
| STW | M | Red |
| STW | L | Red |
| STW | M | Blue |
| STW | L | Blue |

[10] We adapt the term from [14], which investigates direct compression to c-tuples.

[11] Under these assumptions, a *multi-valued decision diagram* (MDD), a more widely known form of a DD [3, 1], can be mapped to a VDD. This is further detailed in [11]

Figure 1 depicts a VDD with set-labeled nodes for Table 1. The HI-links in each path from the root to the sink $\top$ in the VDD in Figure 1 define a c-tuple. The set of all c-tuples that can be formed is disjoint and is a way to represent Table 1 in compressed form. Table 2 lists the two c-tuples needed to represent the 11 variants.

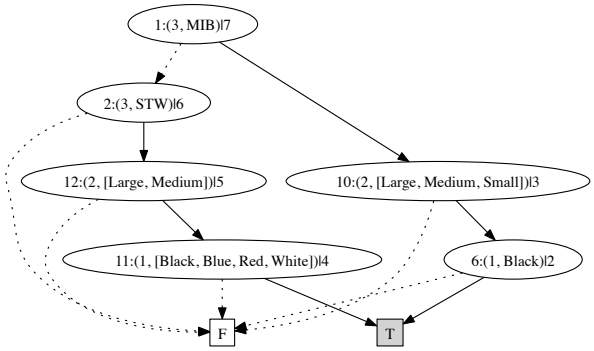


**Figure 1.** VDD of T-shirt with set-labeled nodes

**Table 2.** C-tuples for simple T-shirt

| Imprint | Size | Color |
|---|---|---|
| MIB | S;M;L | Black |
| STW | M;L | Black;White;Red;Blue |

## 4.2 T-Shirt with Infinite Domains

To illustrate the use of infinite sets, we modify the example to allow an arbitrary user-provided image as an imprint on a white T-shirt. An image is identified at runtime via a file name. The file name must refer to a processable graphic, which is taken to mean that only a jpg or a tiff format can be accepted. Hence, the domain is the infinite set of all legal file names that match the regular expression $\langle$*img-filename*$\rangle = *$*.jpg*$| * $*.tiff*.

We also add a property *Scale* to capture a factor to be used to scale the image printed on the T-shirt. For the vintage prints MIB and STW we require $Scale = 1$. For the user-provided images, the scale can be arbitrarily chosen by the user as a floating point number in the range 0.5 to 1 (the interval $[0.5, 1.0]$).

Table 3 lists the c-tuples needed to describe this setting. The product property *Scale* has here been placed as the first property $v_1$. The other properties are now $v_2$ (*Imprint*), $v_3$ (*Size*), and $v_4$ (*Color*).

**Table 3.** C-tuples for simple T-shirt with infinite domains

| Scale | Imprint | Size | Color |
|---|---|---|---|
| 1.0 | MIB | S;M;L | Black |
| 1.0 | STW | M;L | Black;White;Red;Blue |
| $[0.5, 1.0]$ | $\langle$img-filename$\rangle$ | S;M;L | White |

**Table 4.** Split c-tuples for simple T-shirt with infinite domains

| Scale | Imprint | Size | Color |
|---|---|---|---|
| 1.0 | MIB | S;M;L | Black |
| 1.0 | STW | M;L | Black;White;Red;Blue |
| $[1.0]$ | $\langle$img-filename$\rangle$ | S;M;L | White |
| $[0.5, 1.0)$ | $\langle$img-filename$\rangle$ | S;M;L | White |

We now discuss how to construct a VDD with set-labeled nodes for these c-tuples. Figure 2 shows the result[12] .

1. We construct a root node $\nu_1$ labeled $\langle v_1, [1.0]\rangle$ starting with the first c-tuple. This node will be used for both the first and second c-tuples in Table 3.

2. We construct the *l-chain* (chain of *LO*-links) for the root node:
   - We pointed out in Section 3 that nodes linked in an *l-chain* need to have disjoint set labels. This is illustrated here. It will be a problem if we label $\nu_2$ with $[0.5, 1.0]$ (third c-tuple), because then the value $Scale = 1.0$ does not allow deciding uniquely for either $\nu_1$ or $\nu_2$. So we split $[0.5, 1.0]$ into the two disjoint c-tuples.

     $[1.0]$ $\langle$img-filename$\rangle$ $\{S, M, L\}$ $White$
     $[0.5, 1.0)$ $\langle$img-filename$\rangle$ $\{S, M, L\}$ $White$

   - Table 4 shows all the c-tuples to be handled in constructing the VDD. The new third c-tuple is covered by $\nu_1$. We label the second node $\nu_2$, linked from $\nu_1$ via its *LO*-link, with the half-open interval $[0.5, 1.0)$ from the fourth c-tuple. This handles the first column.

3. We next process the rest of the three tuples in Table 4 that start with $C_1 := [1.0]$. We create:
   - $\nu_3$, the target for the *HI*-link of $\nu_1$, labeled with $\langle v_2, MIB\rangle$
   - $\nu_4$, the target for the *LO*-link of $\nu_3$, labeled with $\langle v_2, STW\rangle$
   - $\nu_5$, the target for the *LO*-link of $\nu_4$, labeled with $\langle v_2, \langle$img-filename$\rangle\rangle$

4. It is straightforward to handle the third column for the above three c-tuples: nodes $\nu_3$, $\nu_4$, and $\nu_5$ have their *HI*-links pointing to nodes $\nu_6$, $\nu_7$, and $\nu_8$, respectively with the labels depicted in Figure 2:
   - The first of the c-tuples allows only the color *Black* (node $\nu_9$).
   - The second allows all colors (node $\nu_{10}$), and
   - the third allows only the color *White* (node $\nu_{11}$).

   This completely handles the first three c-tuples.

5. It is now trivial to finish the VDD. Node $\nu_2$ still needs to be processed with respect to the last (fourth) c-tuple. But the columns two to four are identical to those in the third c-tuple. Node $\nu_5$ was already constructed for this.

We note that we skirted the issue that the product property *Imprint* ($v_2$) allows both values from a finite list, e.g. $\{MIB, STW\}$, as well as arbitrary "additional" values ($\langle$img-filename$\rangle$). This is not

[12] To reduce the size needed to display the graph, the terminal sink $\perp$ has been omitted. Conceptually, it terminates all chains of LO-links. Also, the nodes $\nu_1, \nu_2, \ldots \nu_n$ are identified by "n1", "n2", . . . "nn"

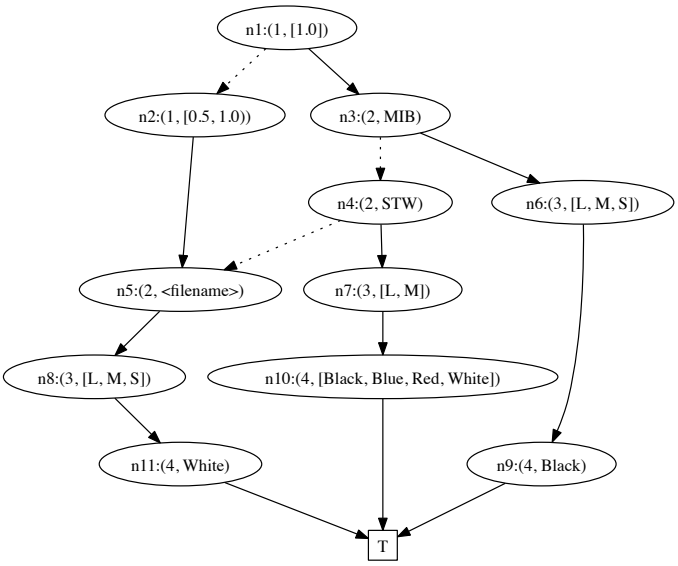


**Figure 2.** VDD of T-shirt with non-finite set-labeled nodes

uncommon in practice where a "standard" solution is modeled with a predefined finite domain, but *additional values* are allowed (see [6]). The sets $\{MIB\}$, $\{STW\}$ and ⟨img-filename⟩ are effectively treated as disjoint by the VDD due to the constructed *l-chain* of nodes $\nu_3$, $\nu_4$, and $\nu_5$). This could be formally ensured by augmenting the QF element ⟨img-filename⟩ to ⟨img-filename⟩ $\cap \neg\{MIB, STW\}$ (see Section 6).

Lastly, we informally discuss some exemplary queries. QF queries are the subject of Section 7. The query to Table 1 for small (*S*) T-Shirts yielded a result set consisting of one r-tuple (the first row). The domains for the three product properties were restricted to $\{MIB\}$, $\{S\}$, and $\{Black\}$. The same query condition against Table 4 yields three c-tuples (the first, third and fourth c-tuple). Each of these c-tuples in the result set must be intersected with the query condition to eliminate the sizes medium (*M*) and large (*L*) that are in the c-tuples but excluded by the query condition. Consequently, the domains for the four product properties are restricted to $[0.5, 1.0]$ $\{MIB, \langle\text{img-filename}\rangle\}$, $\{S\}$, and $\{Black, White\}$[13].

Similarly, the query to Table 1 for *Red* T-Shirts yielded a result set consisting of two r-tuples (the eighth and ninth row). Against Table 4 the result set consists of one (the second) c-tuple. After intersection with the query condition the four product properties are restricted to $[1.0]$ $\{STW\}$, $\{M, L\}$, and $\{Red\}$.

Instead, if the query condition simply specifies a file name for a particular image, *my-img.jpg*, then the last two c-tuples would be the result set. They agree completely except in the first column. As there is no need for the split here (as there was when constructing the original VDD), the two tuples could be combined into one[14]:

$$\langle [0.5, 1.0], \langle\text{img-filename}\rangle, \{S, M, L\}, White\rangle$$

[13] Formed by collecting all symbols occurring for each column and calculating the the *union*. The result of the union of the two intervals is *normalized* (see Section 6)

[14] This reduction is actually required where we want the tuples in a result set to be distinct, i.e. to have been *normalized*.

The result set intersected with the external condition is then:

$$\langle [0.5, 1.0], \{\text{my-img.jpg}\}, \{S, M, L\}, White\rangle$$

If the query formulates the additional restriction $Scale \in [0.25, 0.75]$, then the result set intersected with the external condition is:

$$\langle [0.5, 0.75], \{\text{my-img.jpg}\}, \{S, M, L\}, White\rangle$$

## 5 Excursion on the Construction of VDDs from C-Tuples

A c-tuple $\mathbf{C}$ can be decomposed into its *head* (the first element $C_1$) and its *tail* $\mathbf{T}$, which is also a c-tuple. We denote this by $\mathbf{C} := C_1|\mathbf{T}$:

$$C := \langle C_1, \ldots, C_k\rangle = C_1|\langle C_2, \ldots, C_k\rangle = C_1|\mathbf{T} \qquad (3)$$

When constructing a (partial) VDD from a list of c-tuples $\mathbf{C}_1, \ldots \mathbf{C}_m$ an *l-chain* for the head (root) node is constructed using the first elements $C_{11}, C_{21}, \ldots, C_{m1}$. As discussed in Section 3 and evident from the example in Section 4.2 these elements must be disjoint.

If there are two c-tuples $\mathbf{C}_i, \mathbf{C}_{i'}$ with the same tail, i.e.

$$\mathbf{C}_i = C_{i1}|\mathbf{T} \quad \text{and} \quad \mathbf{C}_{i'1} = C_{i'1}|\mathbf{T}$$

then their first elements must be merged to yield one c-tuple

$$\mathbf{C}' = (C_{i1} \cup C_{i'1})|\mathbf{T}$$

We can ensure disjointness of any other pair of c-tuples $\mathbf{C}_i, \mathbf{C}_{i'}$ with differing tails

$$\mathbf{C}_i = C_{i1}|\mathbf{T}_i \quad \text{and} \quad \mathbf{C}_{i'1} = C_{i'1}|\mathbf{T}_{i'}$$

by replacing them with the three c-tuples $\mathbf{C}_a, \mathbf{C}_b, \mathbf{C}_c$ in (4) (a c-tuple with an empty element is considered empty and can be disregarded):

$$\begin{aligned} \mathbf{C}_a &= (C_{i1} \setminus C_{i'1})|\mathbf{T}_i \\ \mathbf{C}_b &= (C_{i'1} \setminus C_{i1})|\mathbf{T}_{i'} \\ C_c &= (C_{i1} \cap C_{i'1})|(\mathbf{T}_i \cup \mathbf{T}_{i'}) \end{aligned} \qquad (4)$$

As the example in Section 4.2 shows, the c-tuple heads show up directly as labels of set-labeled nodes. We already stated that a set-labeled node labeled with a finite set of symbols (r-symbols or QF-symbols) can be expanded to an l-chain of regular VDD nodes.

## 6 Operations with Non-Finite Elements in C-Tuples

As the discussion in Section 5 and the example in Section 4 make clear, it will be necessary to both split and combine c-tuples when constructing a VDD and result sets. Therefore, we need the following operations on c-tuple elements $C_{ij}, C_{i'j}$ pertaining to the same product property $v_j$:

- set intersection: $C_{ij} \cap C_{i'j}$
- set union: $C_{ij} \cup C_{i'j}$
- negation with respect to the overall domain: $\neg C_{ij} := D_j \setminus C_{ij}$
- set difference: $C_{ij} \setminus C_{i'j} = C_{ij} \cap \neg C_{i'j}$

For finite sets this is a given. For QF-symbols that are used in the labels of VDD nodes, we must ensure that these operations are well defined and fit in our QF framework.

In the following subsections we look at this in detail for the infinite elements we propose to add:

- Real-valued intervals
- Unconstrained countably infinite sets
- Sets of exclusions, particularly *finite* exclusion sets.

Where the domain underlying negation needs to be made clear we will denote negation as:

$$\neg C := \overline{C}^D = D \setminus C$$

## 6.1 Real-Valued Intervals and the *Xnumeric* Datatype

We denote a real-valued interval using conventional mathematical notation, e.g. $[a, b)$ for a half-open interval with a closed lower bound $a$ and an open upper bound $b$. This is the set of all real numbers $x$ such that $x >= a \wedge x < b$. We allow lower and upper infinity, denoted by $-\inf$ and $+\inf$, with open bounds. A single real number $x$ can be encoded as a singleton interval $[x]$. All other interval bounds can be open or closed

We define an *xnumeric* to be a finite list of real-valued intervals representing the union of its elements in a *normalized* form. *Normalized* means that the intervals in the list are disjoint, separable, and in ascending order, e.g. the set of intervals $\{[0.5, 1.0), [1.0]\}$ is disjoint and ascending, but it is not separable. In normalized form it is just $[0.5, 1.0]$. (Remark: The interval $\{[0.5, 1.0), (1.0, 2.0]\}$ is separable and thus normalized, because its two intervals are separated by the "gap" of the singleton interval $[1.0]$.)

For *xnumerics* it is straightforward to ensure normalization. First, any intervals that are non-disjoint or not separable can be merged into one interval. Since the remaining intervals are disjoint, they can be ordered. Hence the union of two xnumeric is just the set union followed by normalization. The intersection is just the list of pairwise intersections. Because the xnumeric is ordered due to normalization, this operation is efficient in the sense that it is not necessary to actually intersect all pairs.

The set union of two intervals is not necessarily again an interval, hence we need the concept of an *xnumeric*.

The *unconstrained xnumeric* is the interval $(-\inf, +\inf)$. The negation of an xnumeric is the set difference to this unconstrained set. It is formed by inverting the finite number of "gaps" between the intervals in the xnumeric. For example

$$\neg\{[0.5, 1.0), (1.0, 2.0]\} = \{(-\inf, 0.5), [1.0], (2.0, +\inf)\}$$

Remark: a finite set of real number values can be represented as an xnumeric using singleton intervals. All interaction with finite sets is covered by the above operations defined for xnumerics.

An xnumeric is a list of QF-symbols (intervals) representing their set union. A set-labeled node for an xnumeric can be expanded to an *l-chain* of nodes with interval labels.

## 6.2 Countably Infinite Sets and Domains

Examples of countably infinite domains are the list of all integers or all strings. This requires that each product property is associated with an immutable datatype. We consider a domain $D$ or any $C \subset D$ to be qualified by a unary *predicate* (condition) that filters out disallowed values at run-time (e.g. a regular expression for a string). Any value fulfilling the predicate (e.g. any string matching the regular expression) is an acceptable value. Examples for qualifying predicates for an integer datatype are: *positive_p*, *even_p* or *odd_p*.[15]

[15] In the absence of more specialized predicates, $\langle true \rangle$ is taken as the default predicate.

A unary predicate can be represented by its name (a symbol), which serves as the QF-symbol identifying it. In the example in Section 4.2, we used the notation $\langle$*img-filename*$\rangle$ to refer to a regular expression for legal file names.

The set operations translate into logical operations for predicates. The union of two infinite sets qualified by predicates $\pi_1$ and $\pi_2$ is just a set qualified with the disjunction $\pi_1 \vee \pi_2$. Similarly, intersection translates to $\pi_1 \wedge \pi_2$, and negation to $\neg\pi_1$.

Again, we require normalization to reduce a complex logical expression by removing any redundant elements. It has yet to be determined what works best in practice here. From a theoretical view, we might require a *disjunctive normal form* (DNF). The overall predicate could then be represented as a list (finite set) of conjunctions. A set-labeled node for such a list can be expanded to an *l-chain* of nodes, as for xnumerics. Each such node would be labeled by a conjunction of predicates, which would be treated as an indivisible QF-symbol.

We must also deal with set unions between finite sets and countably infinite sets. In the example in Section 4, the standard imprints for the T-shirt formed a finite set $\{MIB, STW\}$, but "additional values" were then allowed, which were specified by the QF-symbol $\langle$img-filename$\rangle$. The domain of the property imprint is just the union of these sets. Generally, the domain $D$ for a product property with a *non-xnumeric* datatype is $D = \{F, \pi\} := F \cup \pi$, where $F$ is a finite set of values, $\pi$ a predicate representing an countable infinite set, and both $F$ and $\pi$ respect the datatype assigned to the product property.[16]

The set-operations then become:

- set intersection: $\{F, \langle\pi\rangle\} \cap \{F', \langle\pi'\rangle\} = \{F \cap F', \langle\pi \wedge \pi'\rangle\}$
- set union: $\{F, \langle\pi\rangle\} \cup \{F', \langle\pi'\rangle\} = \{F \cup F', \langle\pi \vee \pi'\rangle\}$
- negation: $\neg\{F, \langle\pi\rangle\} := \overline{F}^{\langle\pi\rangle} \cap \neg\langle\pi\rangle$
- set difference: $\{F, \langle\pi\rangle\} \setminus \{F', \langle\pi'\rangle\} = \{F'', \neg F', \langle\pi\rangle \setminus \langle\pi'\rangle\}$
  - where $F''$ is the finite set $F \setminus F' \cup F \setminus \langle\pi'\rangle$, and
  - the finite set $\neg F' = \overline{F'}^{\langle\pi\rangle}$ is an *exclusion set* of all values in $F'$ that lie in $\langle\pi\rangle$ (see Section 6.3).

## 6.3 Exclusions and Exclusion Sets

An *exclusion* of a value $x$ from a property domain $D$ is a way of stating $D \setminus \{x\}$. It means that $x$ is considered to be invalid, which we will denote by $\neg x$. For real-valued domains, exclusions can be directly formulated as xnumerics, e.g., $\{(-\inf, x)(x, +\inf)\}$ would exclude the real number $x$. For a finite domain or an xnumeric domain, we can simply positively represent the set $D \setminus \{x\}$. For a countably infinite domain, we need further expressiveness. Given a countably infinite domain $D$ for a product property and a finite set of values $E \subset D$, we introduce an *exclusion set* $\neg E := \overline{E}^D := D \setminus E$. An exclusion set $\neg E$ can be merged with a unary predicate $\pi$ by removing any values from $E$ that do not satisfy the predicate $\pi$, i.e. $\neg E \cap \langle\pi\rangle \subset \neg E$ is a reduced exclusion set. In order to keep the exposition simple, we will ignore this reduction and denote $\neg E \cap \langle\pi\rangle$ also by $\neg E$.

For two exclusion sets $\neg E, \neg E'$, the required set operations are inverted:

- set intersection: $\neg E \cap \neg E' = \neg(E \cup E')$
- set union: $\neg E \cup \neg E' = \neg(E \cap E')$
- negation: $\neg\neg E = D \setminus (D \setminus \neg E) = E$

[16] Ideally, $F$ and $\pi$ will be disjoint. Either $F$ or $\pi$ can be empty. We define the predicate $\langle$*false*$\rangle$ to represent the empty set.

- set difference: $\neg E \setminus \neg E' = E' \setminus E$

Finite exclusion sets are needed in order to meet our requirements of negation of finite sets against infinite domains. The concept can also be extended to infinite exclusion sets. Indeed, a negated unary predicate corresponds to such a set. For example, if the set of all prime numbers is represented by the unary predicate $\langle prime\_p \rangle$, the $\neg\langle prime\_p \rangle$ represents exclusion of all prime integers.

In either case, a reference to an exclusion set is treated as a QF-symbol. For example, for a predicate $\pi$, $\neg\pi$ is the symbol representing the exclusion of all values in $\pi$.

## 7 Queries on Quasi-Finite VDDs

In the classic finite case, the result set $\mathfrak{R}$ of the query (1) is a finite set of *r-tuples*. In the QF framework, it is a finite set of *QF-tuples* that may contain both value symbols and QF-symbols. A QF-symbol in the result set must be specialized to conform to the the query condition, e.g. by set intersection with the query condition. Problem solving (PS) must expect the remaining degree of non-determinism.

The query (2) contains the keyword *DISTINCT*. This means any duplicates must be removed from the result set for the particular column (property). We see replacing QF-symbols by their *normalized union* akin to removing duplicates. Therefore, the symbols in the result set, both QF-symbols and r-symbols, must be replaced by their normalized union, which ensures also that remaining symbols are pairwise disjoint.

## 8 Constraint Processing with Quasi-Finite Symbols

### 8.1 Specialization Relations

Given two QF-symbols $\phi_1, \phi_2$ for the same CSP variable, we regard $\phi_2$ to be more *special* than $\phi_1$ if $\phi_2$ denotes a subset of $\phi_1$. This leads to a partial ordering (PO) of the symbols that occur in the variant tables, which we call a *specialization relation*, introduced and motivated for another context in [9]. Generally, a *specialization relation* on a set of facts expresses specificity of meaning, characterized by the following three properties:

- *Problem solving* (PS) need not consider an otherwise valid fact in the presence of a more special one (*procedural-subsumption property*). This property requires the acquiescence of PS.
- A fact is logically implied by any of its specializations (*semantic-compatibility property*).
- Negation inverts specialization (symmetry-under-negation property).

The facts we deal with in this paper are assignments of r-symbols and QF-symbols to a CSP variable. The PS we consider consists of queries to the table and constraint processing, particularly local propagation of constraints. From the perspective of queries we additionally need to be able to aggregate the result sets into a normalized form, e.g. delete duplicate r-symbols, replace two QF-symbols by a more general one representing their union, etc. We have shown in Section 6 that the QF-symbols we primarily envision meet these requirements.

We can also turn the reasoning around and define a PO of symbols pertaining to the same property domain as a specialization relation if we can show that it has the above properties and if we also guarantee the following:

- There is a unique symbol $\perp$ (*false*) that is a special of all other symbols. This is a QF-symbol for the empty set.
- For any two symbols in the PO, there exists a unique symbol for a *greatest common successor/special* (the "*intersection*").
- For any two symbols in the PO, there exists a unique symbol for a *least common predecessor/general* (the "*normalized union*")[17].
- There is a unique top-level symbol $\Omega$ that represents the entire domain. For any symbol $\phi$ in the PO, there exists a symbol $\neg\phi$ in the PO, such that the least common predecessor of $\phi$ and $\neg\phi$ is $\Omega$ and the greatest common successor is $\perp$. $\neg\phi$ denotes the negation of $\phi$.[18]

For example, we could arrange images in a PO and declare it a specialization relation, paying some attention to fulfill the requirements in the spirit of the intended PS.

Specialization relations provide some conceptual and practical benefits:

- They can be pre-calculated and stored in a graph. This may be more performant than calculating intersections and unions on the fly.
- They provide a general concept to adapt PS to QF-symbols: PS must simply be prepared to specialize the assignment of a QF-symbol to a CSP variable.
- They generalize to other objects, e.g. shapes, images, taxonomies, etc.

### 8.2 Local Propagation with QF-Symbols

If a product model contains multiple constraints, *local propagation* [4] can be used to restrict the domains of the product properties to a state of *arc consistency*. Any domain restriction of a product property is propagated to all constraints that reference the same product property. The process continues until no further restrictions are possible. For a tabular constraint, the query (2) can be used to determine the domain restrictions, which are then propagated (see Section 7).

The QF framework fits nicely in this scheme. A c-tuple comprised of finite sets of symbols that may include the normalized union of QF-symbols may be used as a query condition. The domain restrictions that result from the query (2) with this query condition may again contain the normalized union of QF-symbols. The c-tuple formed from the resulting domain restriction for each column can be smoothly propagated to other constraints.

If the product model is entirely made-up of tabular constraints, the local propagation of QF-symbols is covered by our approach. It is also straightforward to include constraints representing numeric linear (in)equalities when propagating real-valued intervals.[19] Extending the propagation of QF-symbols yet further is a topic of future work.

### 8.3 General Constraint Solving

Extending existing problem solvers to deal with QF-symbols, will require an analysis of the particular methods employed. However, a

[17] We had noted that the union of two QF-symbols need not itself be a QF-symbol. Here, however, it is an advantage to be able to have a least common predecessor/general representing the *union* as part of the PO. A more detailed treatment of specialization relations is deferred to a discussion using practical examples when they arise.

[18] We need "negation" primarily to ensure a "set-difference" operation to be able to split two symbols into a disjoint triple of symbols as in (4) in Section 5.

[19] This is implemented in the SAP product configurators ([6]).

main common idea is that constraint problem solving will make use of the concept of *specialization relations*. Instead of exploring the validity of a simple value assignment, an assignment of a variable to a QF-symbol can be specialized. When considering such an assignment and a constraint on the same variable, the greatest common special must be substituted in the assignment. A forced specialization to the empty set would invalidate an assignment. The solutions found by constraint solving may contain (specialized) QF-symbols, i.e. exhibit a degree of non-determinism that cannot be avoided and is to be expected.

## 9 Indeterminism in Variants and Sub-Variants

A product variant is classically defined as an r-tuple, a value assignment to each of its product properties, but this is neither ideal nor sufficient in practice. Some degree of indeterminism in a variant is needed when a variant is to be further specialized in a later business process (e.g., at the customer's site). For example, a pump may be sold with a connection that fits several different sizes of hoses. The end customer may have to make a manual adjustment for the particular hose they want to attach by cutting off a part of the provided connector. The pump being sold to the customer by the business is a variant of their MC "pump" product that allows further individualization at the customer's site.

The number of sub-variants can be infinite, e.g. for the frequency a radio receiver may be tuned to. As built, the property *Frequency* would be described by a list of real-valued intervals for possible reception bands, e.g. $\{[7.2, 7.45], [9.4, 9.9], [11.6, 12.1]\}$ (MHz).

Allowing a variant to be defined by a c-tuple solves the above problem. However, c-tuples that define variants must be distinguished from those that are merely the by-product of compression. This would be an open MC business topic.

## 10 Summary and Outlook

This work extends a common theme: that data in tabular form is not only natural for modeling variants, but also a natural, non-proprietary medium for communicating between interrelated business processes within an enterprise, as well as between enterprises. Compression of tables is essential in MC for letting variant tables scale with a growing number of choices, which production technology now enables [13]. C-tuples are a transparent yet powerful form of compression that is transparent and upon which non-proprietary exchange formats can be based. This is addressed in other work, e.g. [13]. Here, we propose to add to the expressivity of variant tables by presenting a *quasi-finite* (QF) framework that allows dealing with infinite sets of choices within the tabular paradigm.

We believe the QF framework presented here meets these expectations. The main idea is that problem solving will deal with the QF-symbols representing infinite sets via *specialization relations*. Instead of exploring the validity of a value assignment of a value (feature) to a variable (product property), the assignment to a QF-symbol can be specialized. A forced specialization to the empty set would invalidate an assignment.

The discussed techniques involving c-tuples and VDDs using QF-symbols has not yet been deployed in practice. The individual ingredients: c-tuples, VDDs, and the management of partial orders (POs) for specialization relations have all been applied with positive results. As already mentioned, the question of how to define practical normalization of predicates is open. However, we believe that the primary open issue is to verify that it actually meets the expectations of MC business. This also includes evaluating the need of integrating with other types of constraints, such as linear (in)equality constraints, and the business value of indeterminism in product variants.

## ACKNOWLEDGEMENTS

I would like to thank my daughter Laura and the reviewers for their comments, which helped improve this paper considerably.

## REFERENCES

[1] Jérôme Amilhastre, Hélène Fargier, Alexandre Niveau, and Cédric Pralet, 'Compiling csps: A complexity map of (non-deterministic) multivalued decision diagrams', *International Journal on Artificial Intelligence Tools*, **23**(4), (2014).

[2] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann, 'A constraint store based on multivalued decision diagrams', In Bessiere [5], pp. 118–132.

[3] Rüdiger Berndt, Peter Bazan, Kai-Steffen Jens Hielscher, Reinhard German, and Martin Lukasiewycz, 'Multi-valued decision diagrams for the verification of consistency in automotive product data', in *2012 12th International Conference on Quality Software, Xi'an, Shaanxi, China, August 27-29, 2012*, eds., Antony Tang and Henry Muccini, pp. 189–192. IEEE, (2012).

[4] C. Bessiere, 'Constraint propagation', in *Handbook of Constraint Programming*, eds., F. Rossi, P. van Beek, and T. Walsh, chapter 3, Elsevier, (2006).

[5] Christian Bessiere, ed. *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*. Springer, 2007.

[6] U. Blumöhr, M. Münch, and M. Ukalovic, *Variant Configuration with SAP, second edition*, SAP Press, Galileo Press, 2012.

[7] A. Haag, 'Chapter 27 - Product Configuration in SAP: A Retrospective', in *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 319 – 337, Morgan Kaufmann, Boston, (2014).

[8] Albert Haag, 'Konzepte zur praktischen handhabbarkeit einer atms-basierten problemlösung', in *Das PLAKON-Buch, Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen*, eds., Roman Cunis, Andreas Günter, and Helmut Strecker, volume 266 of *Informatik-Fachberichte*, 212–237, Springer, (1991).

[9] Albert Haag, *The ATMS - an assumption based problem solving architecture utilizing specialization relations*, Ph.D. dissertation, Kaiserslautern University of Technology, Germany, 1995.

[10] Albert Haag, 'Column oriented compilation of variant tables', in *Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015.*, eds., Juha Tiihonen, Andreas A. Falkner, and Tomas Axling, volume 1453 of *CEUR Workshop Proceedings*, pp. 89–96. CEUR-WS.org, (2015).

[11] Albert Haag, 'Managing variants of a personalized product', *Journal of Intelligent Information Systems*, 1–28, (2016).

[12] Albert Haag, 'Assessing the complexity expressed in a variant table', in *Proceedings of the 19th International Configuration Workshop, La Defense, France, September 14-15, 2017.*, pp. 20–27, (2017).

[13] Albert Haag and Laura Haag, 'Empowering the use of variant tables in mass customization', in *Proceedings of the MCP-CE 2018 conference, Novi Sad, Serbia, September 19-21, 2018.*, p. (Submitted), (2018).

[14] G. Katsirelos and T. Walsh, 'A compression algorithm for large arity extensional constraints', In Bessiere [5], pp. 379–393.

# Software Configuration Diagnosis – A Survey of Existing Methods and Open Challenges

**Artur Andrzejak**[1] and **Gerhard Friedrich**[2] and **Franz Wotawa**[3]

**Abstract.** As software systems become more complex and feature-rich, configuration mechanisms are needed to adapt them to different execution environments and usage profiles. As a consequence, failures due to erroneous configuration settings are becoming more common, calling for effective mechanisms for diagnosis, repair, and prevention of such issues. In this paper, we survey approaches for diagnosing software configuration errors, methods for debugging these errors, and techniques for testing against such issues. In addition, we outline current challenges of isolating and fixing faults in configuration settings, including improving fault localization, handling the case of multi-stack systems, and configuration verification at runtime.

## 1 Introduction

Tackling software configuration errors is recognized as an important research problem which has been investigated by many groups from academia and industry, e.g., see [51]. In a recent study [52], the authors report empirical findings on the impact of configuration errors in practice. In particular, a study of over 500 real-world configuration issues revealed that this type of problems constituted the largest percentage (31%) of high-severity support requests. Moreover, a significant portion of these issues (16% to 47%) rendered systems fully unavailable or caused severe performance degradation. Also other studies [30] and incident reports [5] confirm that detecting and correcting configuration errors in software is of a great importance for practical applications.

In this paper, we focus on providing an overview of current research in the area of software configuration diagnosis comprising fault detection, fault localization, and correction. Besides discussing research articles dealing with software configure errors, we further discuss open issues and challenges that are worth being tackled in future research activities. While the excellent survey [51] has a broader scope and also includes aspects such as configuration-free/easy-to-configure systems, hardening against configuration errors, automating deployment and monitoring etc., we consider in this paper primarily diagnosis aspects. We also cover the most recent state-of-the-art work like diagnosing cross-stack configuration errors [32]. In summary, this survey attempts to offer a compact and focused introduction to this research area, thus serving as a good starting point for further contributions.

Although, there has been work also dealing with configurations and configuration errors for systems comprising hardware and software, we focus on methods and tools that have been developed within the area of software configuration. Dealing with software configuration only allows for extracting and straightforwardly using information from programs, which would be hardly obtained when considering hardware. As a consequence, there are many approaches that work exclusively in the software configuration domain. Nevertheless, there are also approaches that can be generalized to serve diagnosis of system configuration as well. Especially, when it comes to large software comprising million lines of source code and also to cases where source code is not available, approaches have to follow a more black-box oriented approach. This approach also enables diagnosis in case of hardware or systems in general where hard- and software is investigated.

In more detail, given a program, its configuration parameters (or settings), and an execution environment, a *software configuration error* comes forward when the parameters assume incorrect values. The configuration parameters might specify multiple aspects of system behavior, including adaptation to execution environment (paths, network settings, ..), functionality (enabled/disabled components, logging, ...), performance and resource policies (cache sizes, number of threads, ..), security settings, and others. Consequently, erroneous configuration settings can cause failures of multiple types: complete crashes, partially disabled functionality, performance issues, inappropriate resource usage, or security threads. A frequent scenario of a configuration error are parameter values which do not fit to the specific execution environment. For example, we specified a path to a working directory of the application but the user executing the program do not have write access to this directory, causing the program to crash (or at least to terminate with an exception).

In the context of this survey, we consider the *configuration error diagnosis problem* in its most general form: detecting the root causes, i.e. isolating the configuration parameters with inappropriate values, and providing means for repair in terms of identifying correct values or value ranges for these parameters (or adapting the execution environment). This definition implies that we do not target diagnosis of "traditional" software bugs, since we assume that a repair is possible without code changes. Note that it might be difficult to decide whether a failure should be attributed to a configuration problem or a software bug, and this challenge remains one of the open issues (see Section 3). For example, if a failure-triggering sequence of statements in a faulty program is executed only because of a certain parameter setting, the subsequent failure might appear to be caused by a configuration error.

We organize this paper as follows: We first discuss in Section 2 previous research works dealing with software configuration diagnosis. In the following Section 3 we present open research challenges that have not been tackled so far. We discuss threats to validity in

[1] Heidelberg University, Germany, email: artur.andrzejak@informatik.uni-heidelberg.de
[2] University Klagenfurt, Austria, email: Gerhard.Friedrich@aau.at
[3] TU Graz, Institute for Software Technology, Austria, email: wotawa@ist.tugraz.at

Sec. 4. Finally, we summarize the content and the findings of this paper (Section 5).

## 2 Previous Work on Software Configuration Diagnosis

In this section, we discuss research work that has been published in the area of software configuration diagnosis. We obtained the papers searching relevant digital libraries from IEEE and ACM. We further focussed on the most recent work in this area not older than 10 years. Hence, we do not claim the survey to comprise all papers in the context of software configuration errors (for a more comprehensive collection see [51]). However, the presented papers are intended to give an overview of the current research directions in software configuration diagnosis and methods and techniques used for this purpose.

In order to present the discussed papers in an accessible way, we classify the paper accordingly to the following categories: (i) diagnosing single-layer configuration errors, (ii) diagnosing cross-stack configuration errors, (iii) diagnosing using configuration knowledge, and (iv) other aspects of software configuration diagnosis. *Single-layer configuration errors* are errors found in one-component applications like MySQL, Hive, or Spark. Typically, such applications have one common configuration file/database and are developed as an integral project. *Cross-stack configuration errors* occur in multi-component applications or software stacks like LAMP (Linux, Apache Web Server, MySQL, PHP, Wordpress/Drupal), J2EE, or MEAN.

The rational behind these categories is the following. Most previous work is available for diagnosing single-layer configuration errors and this case offers an opportunity for an overview of existing diagnosis approaches. Diagnosis of cross-stack configuration errors pose additional challenges. In some cases, the source code of stack components might not be available, precluding usage of general program analysis techniques. More frequently, cross-stack configuration errors are frequently caused by a mismatch between the configuration settings within separate components [32, 33]. To diagnose such issues, knowledge about the interactions between the components is needed.

In case of the availability of formal knowledge about configurations, i.e., configuration rules or constraints, diagnosis can be performed using this knowledge. Such formal knowledge bases may be applicable for single-layer or cross-stack applications.

Finally, there are other aspects that cannot be assigned to one of the former categories, for example testing configurable systems or optimization of software based on configuration parameters.

### 2.1 Diagnosing Single-Layer Configuration Errors

Single-layer programs are typically written in a single programming language and often the source code is available. Hence, static and dynamic program analysis techniques can be applied to obtain a mapping from configuration options to code regions. This information can be exploited for localizing the root cause behind configuration errors. Consequently, a lot of approaches for diagnosis configuration errors in such programs have been proposed.

***Linking configuration options and code regions.*** Approaches in this group attempt to find a correspondence between a configuration option and code regions impacted by this option. Frequently, such techniques exploit static [43] or dynamic program slicing [14]. In program slicing, one attempts to find the set of all code locations which might influence a target statement (so-called seed), or all code locations which might be influenced by a seed statement. Hence, there approaches are mainly applicable in the software configuration setting and may not be generalizable to deal with hardware configuration diagnosis.

ConfAnalyzer [29] builds a map from each program point to the options that might cause an error at that point by static data-flow analysis. For diagnosis, it treats a configuration option as the root cause if its value flows into the crashing point. The approach does not require from users to install or use additional tools, but it can use logs and stack traces to reduce the rate of false positives.

ConfDiagnoser [57, 56] uses static analysis, dynamic profiling, and statistical analysis to link the undesired behavior that are represented by predicates to configuration options. When these predicates indicate behavior deviating from the one known for correct profiles, ConfDiagnoser lists the relevant configuration options as suspects.

Work [58] presents a technique and a tool to troubleshoot configuration errors caused by software evolution. The approach uses dynamic profiling, execution trace comparison, and static analysis to link the undesired behavior to its root cause - a configuration option which needs to be changed in the new software version.

ConfDoctor [7] is an approach based on static analysis to diagnose configuration defects. It does not require users to execute an instrumented program or to reproduce errors, which is an essential advantage compared to previous approaches. The only run-time information required is the stack trace of a failure. An evaluation on JChord, Randoop, Hadoop, and Hbase shows that the approach could successfully diagnose 27 out of 29 errors, with 20 of them ranked first.

Authors of [25] propose a lightweight dynamic analysis technique that automatically discovers a program's interactions, i.e., logical formulae that give developers information about how a system's configuration option settings map to particular code coverage. It is evaluated on 29 programs spanning five languages and could find precise interactions based on a very small fraction of the number of possible configurations.

***Data flow analysis.*** ConfAid [3] applies dynamic information flow analysis techniques to track tokens from specified "configuration sources" and analyze dependencies between the tokens and the error symptoms, pinpointing which tokens are root causes.

Sherlog [53] uses static analysis to infer control and data information in case of a failure. It analyses source code by exploiting information from run-time logs and computes what must or may have happened during the failed run. One deficiency of this tool is that it may require guidance from developers about which function should be symbolically executed.

Paper [17] introduces Lotrack, an extended static taint analysis approach and tool to automatically track configuration options. It derives a configuration map that explains for each code fragment under which configurations it may be executed.

***Supervised learning approaches.*** Relatively few authors propose to use machine learning approaches based on supervised learning (i.e. mainly classification). This can be explained by the fact that it is difficult to obtain or generate training data with appropriate structure and in sufficient amount. Similarly to the challenges of mutation testing, if training samples are generated, faults injected in the configuration files might not trigger a failure or have unrealistic properties. Also, since a configuration file might contain hundreds of options, a training set is likely to containt only few faulty cases per option, giving rise to the unbalanced class problem.

Authors of [41] use machine learning to predict whether a configuration error is responsible for a failure and if yes, what is the category of the error. To obtain training data, faults are injected into configuration files and the resulting error category is manually labeled.

Work [38] exploits statistical decision tree analysis to determine possible misconfigurations in data center environments. The authors further improve the accuracy of this approach via a pattern modification method.

***Replay-based techniques.*** One category of well-known tools [44, 37, 20] are the replay-based diagnosis techniques. They treat the system as a black box to automatically run the system with possible configurations values without damaging the rest of the system until fixing the misconfiguration. This class of techniques relies on having a working configuration. Otherwise, it can not be applied. Besides, they require users with more domain knowledge.

***Signature-based approaches.*** Another family of tools mine a large amount of configuration data from different instances to infer rules about options and use these rules to identify software misconfigurations.

EnCore [55] and CODE [54] belong to this category of work. EnCore takes into account the interaction between the configuration settings and the executing environment, as well as the correlations between configuration entries. It learns configuration rules from a given set of sample configurations and pinpoints configuration anomalies based on these rules.

Analogously, some tools such as Strider [42] or PeerPressure [40] adopt statistical techniques to compare values of configuration options in a problematic system with those in other systems to infer the root cause of a failure. All these techniques require substantial effort to collect the baseline data.

## 2.2 Diagnosing cross-stack configuration errors

Configuration options in multi-layer architectures (e.g., LAMP, J2EE, or MEAN "software stacks") might easily contradict each other or be hard to trace to each other. Therefore, configuration error diagnosis in such architectures is particularly challenging [51]. On the other hand, so far there are very few research approaches or tools targeting this scenario [33].

Sayagh and Adams [32] conducted an empirical study on multi-layer configuration options across Wordpress (WP) plugins, WP, and the PHP engine. They discover a large and increasing number of configuration options used by WP and its plugins. In addition, over 85% of these options are used by at least two plugins at the same time.

Sayagh *et al.* [33] perform a qualitative analysis of over 1,000 configuration errors to understand their impact and complexity. Based on this data they develop a slicing-based approach to identify error-inducing configuration options in heterogeneous software stacks. So far it is the only approach which attempts to provide a complete, end-to-end process for diagnosing cross-stack configuration errors.

Work [4] focuses on finding configuration inconsistencies between layers in complex, multi-component software. The proposed technique (based on static analysis) can handle software that is written in multiple programming languages and has a complex preference structure.

In [31] the authors target the identification of configuration dependencies in multi-tiered enterprise applications. It provides a method for analyzing existing deployments to infer the configuration dependencies in a probabilistic sense. This yields rank-ordered list of dependencies so that administrators can consult it and systematically identify the true dependencies.

Authors of [12] attempt to quantify the challenges that configurability of complex, multi-component systems creates for software testing and debugging. It analyzes a highly-configurable industrial application and two open source applications. They notice that all three applications consist of multiple programming languages, limiting the applicability of static analysis. Furthermore, they find out that there many access points and methods to modify configurations, and that the configuration state of an application on failure cannot be determined only from persistent data.

## 2.3 Rules, Constraints and Fixing their Violations

Once configuration knowledge can be described using constraints or rules they can be used for diagnosis as well. The use of such knowledge is neither restricted to single-layer nor cross-stack applications in general. Hence, methods and techniques based on rules and constraints, which can also be seen as models of the applications, would provide a more general account to solve the software configuration error problem. In this section, we distinguish methods for learning knowledge, fixing violations, and inconsistency detection between different software artifacts.

***Learning constraints and rules.*** Several existing approaches extract configuration models [42, 40, 54, 50, 55] and leverage them for configuration debugging, mainly via detecting value anomalies and rule violations.

The categories of extracted data constituting the models typically include the primitive and semantic *data types* of configuration options (e.g., integer, file path, port number, URL), the *value ranges* of options (minimum and maximum integer values or a list of acceptable values), the *control dependencies* (i.e., usage of parameter $Q$ relies on the setting of another parameter $P$), and *value relationships* (e.g., value of parameter $S$ should be greater than that of parameter $T$). EnCore [55] additionally considers the properties of the execution environment as a part of their models.

CODE [54] takes a unique approach and uses dynamic execution information as the model content, namely sequences of (Windows) registry accesses and derived rules. Using these rules for efficient filtering of even large lists of events, CODE can detect not only configuration errors but also deviant program executions. It requires no

source code, application-specific semantics, or heavyweight program analysis.

SPEX [50] analyzes source code to infer configuration option constraints and use these constraints to diagnose software misconfigurations, to expose misconfiguration vulnerabilities, and to detect error-prone configuration design and handling.

***Build-time configuration settings.*** Another category of work addresses configurations and their constraints used at compilation and build time. Such configurations determine whether certain product features (e.g. logging, debugging) are activated, or even which software components are included in the shipped product. The later aspect is relevant e.g., for software product lines.

Works [22], [23] propose a static analysis approach to extract (build-time) configuration constraints from C code. Despite of its simplicity, it has high precision (77% - 93% in the studied systems) and can recover 28% of existing constraints. A further study of the authors reveals that configuration constraints enforce correct runtime behavior, improve users' configuration experience, and prevent corner cases.

***Fixing violations of configuration constraints.*** The problem of fixing a configuration that violates one or more constraints is addressed in [47, 48]. The authors introduce to this purpose the concept of a range fix, which specifies the options to change the ranges of values for these options. They also design and evaluate an algorithm that automatically generates range fixes for a violated constraint. Empirical studies shows that the range fix approach provides mostly simple yet complete sets of fixes and has a moderate running time in the order of seconds.

Configurable software (e.g., Linux OS, eCos) can have very high number of options (variables) and constraints. E.g., Linux has over 6,000 variables and 10,000 constraints; eCos has over 1,000 variables and 1,000 constraints. Such systems typically use variability modeling languages and configuration tools (called *configurators*). Examples of variability languages include Linux Kconfig, eCos CDL, and feature models. With variability modeling languages and configurators, errors can be detected early, but users still have to resolve the errors, which is also not an easy task: the constraints in variability models can be very complex and highly interconnected. Therefore, researchers have proposed automated approaches that suggest a list of fixes for an error. A fix is a set of changes that, when performed on the configuration, resolve the current error. However, the recommended fixes in these approaches are sometimes large in number and size. For example, fix lists for eCos configurations contain up to nine fixes, and some fixes change up to nine variables.

In this context, work [39] proposes a method to reduce the size and complexity of error fixes by introducing a concept of *dynamic priorities*. The basic idea is to first generate one fix and then to gradually reach the desirable state based on user feedback. To this end, the approach (1) automatically translates user feedback into a set of implicit priority levels on variables, using five priority assignment and adjustment strategies and (2) efficiently identifies potentially desirable fixes that change only the variables with low priorities.

**Detecting inconsistencies between code, documentation, and configuration files.** Configuration options are widely used for customizing the behavior and initial settings of software applications, server processes, and operating systems. Their distinctive property is that each option is processed, defined, and described in different parts of a software project - namely in code, in configuration file, and in documentation. This creates a challenge for maintaining project consistency as it evolves. It also promotes inconsistencies leading to misconfiguration issues in production scenarios.

Confalyzer [30] uses static analysis to extract a list of configuration option from source code and from associated options documentation. Confalyzer first marks configuration APIs in the configuration classes. Then it identifies calls to these APIs in the program by building a call graph and obtains option names by reading values of parameters of these calls.

PrefFinder [11] proposed by Jin *et al*., uses static analysis and dynamic analysis techniques to extract configuration options and stores them in a database for query and use.

The SCIC approach [4] exploits Confalyzer to implement the functionality of extracting configuration options in the key-value model and the tree-structured model.

Work [6] proposes an approach for detection of inconsistencies between source code and documentation based on static analysis. It identifies source code locations where options are read and for each such location retrieves the name of the option. Inconsistencies are then detected by comparing the results against the option names listed in documentation.

## 2.4 Other Aspects

There are other papers dealing with diagnosis of software configuration errors not falling into the previous categories like testing, end-user support and performance optimization, which we discuss in this subsection.

***Testing of highly configurable systems.*** Paper [18] presents an initial study on the potential of using statistical testing techniques for improving the efficiency of test selection for configurable software. The study aims to answer whether statistical testing can reduce the effort of localizing the most critical software faults, seen from user perspective.

Authors of [19] analyze program traces to characterize and identify where interactions occur on control flow and data. They find that the essential configuration complexity of these programs is indeed much lower than the combinatorial explosion of the configuration space indicates.

Work [36] proposes S-SPLat, a technique that combines heuristic sampling with symbolic search to explore enormous space of configurations for testing of software product lines.

A more general approach for testing configurable systems including software is combinatorial testing [15, 16]. There the underlying assumption is that it is not necessarily one configuration parameter that reveals a fault but a certain combination of parameters. Combinatorial testing assures to compute all combinations for any arbitrary subset of configuration parameters of arity $k$. In the context of combinatorial testing, the resulting test suite is said being of strength $k$. There are many algorithms and tools for combinatorial testing [13]. For a survey on combinatorial testing we refer the interested user to [26].

***Configuration and debugging support for end-users.*** A technique to detect inadequate (i.e., missing or ambiguous) diagnostic messages for configuration errors issued by a configurable software system is proposed in [59]. It injects configuration errors and uses natural language processing to analyze the resulting diagnostic messages. It then identifies messages which might be unhelpful in diagnosis or even negatively impact this process.

Authors of [49] study configuration settings of real-world users from multiple projects and reveal patterns of unnecessary complexity in configuration design. The authors also provide a few guidelines to reduce the configuration space. Finally, the existing configuration navigation methods are studied in terms of their effectiveness in dealing with the over-designed configuration.

Work [28] introduces ConfSeer, a system which recommends to users suitable knowledge base articles which are likely to describe user's current configuration problem and its fix. To this end, ConfSeer takes the snapshots of configuration files from a user machine as input, then extracts the configuration parameter names and value settings from the snapshots and matches them against a large set of KB articles. If a match is found, ConfSeer pinpoints the configuration error with its matching KB article. The described system powers the recommendation engine behind Microsoft Operations Management Suite.

***Optimizing performance via configuration settings.*** In [24], a rank-based approach to efficient creation of performance models is introduced. Such models can be exploited for finding an optimally performing configuration of a software system.

Authors of [10] conducted an empirical study on four popular software systems by varying software configurations and environmental conditions, to identify the key knowledge pieces that can be exploited for transfer learning for constructing performance models of configurable software systems.

Paper [35] proposes a multi-objective evolutionary algorithm to find the optimal solutions and addresses the configuration optimization problem for software product lines.

Finally, the work described in [27] employs random sampling and recursive search in a configuration space to find optimally performing configurations for an anticipated workload in software product lines.

## 2.5 Survey Summary

There are lots of papers dealing with configuration diagnosis of single layer applications often employing program analysis techniques but also making use of machine learning or replay methods. In case of more complicated applications comprising interacting and configurable software components there have been less papers dealing with concrete solutions. One approach that can be used in both cases of software is to make use of formalized knowledge about configurations, i.e., the configuration parameters, their domains, and rules specifying limitations and relationships among parameters. It would be interesting to investigate whether classical approaches to diagnosis of knowledge-bases like [8, 45, 9, 34] can also be successfully applied for configuration diagnosis. Other aspects, discussed in this section include testing configurations, end-user support, and performance optimization.

# 3 Challenges in Configuration Diagnosis

Based on the survey of papers presented in the previous section, we are able to identify several still open challenges. A general challenge that immediately arises is to distinguish whether an application failure is due to a fault in the configuration setup or code defect in the program. This is a common problem when applying configuration debugging tools, which usually assumes a certain cause. If we want to come up with a general approach for software configuration diagnosis, we have to adapt diagnosis to identify the underlying root cause.

A method that is able to separate these causes would take the current configuration, the program, the description of the execution environment, and the passing/failing tests as input. Based on these inputs the possible causes of a failure are provided as output. In order to come up with such an approach, it is necessary to have a close look at various configuration diagnosis problems, given consequently raise to the another challenge, i.e., providing an open repository of various configuration diagnosis problems that can be accessed by researchers in this field.

Such a general repository for software configuration diagnosis should include a larger set of different programs from single-layer to cross-stack applications together with configuration errors coming from different sources, test suites, and ideally also configuration knowledge bases. The repository should cover programs of different sizes and from different domains capturing currently available software to allow comparing different configuration diagnosis methods and techniques.

Besides these two general challenges, there are other challenges that are more specific to the applications (single-layer versus cross-stack) or the tasks to be tackled (i.e., fault localization and repair versus fault detection). In the following, we illustrate some of these more specific challenges in detail.

**Diagnosis of single-layer applications** Despite the fact that there have been various methods already published in this domain, there are still some open issues.

- Transfer techniques from functional fault localization: In case of software debugging, there are various methods available going beyond program analysis including spectrum-based fault localization [1, 2] among others. In this approach, code regions are ranked (essentially) according to the number of times there are executed by passing or by failing tests (intuition: if a code line is executed primarily by failing tests, it is more likely to contribute to a failure). For a detailed look at current debugging techniques we refer the interested reader to Wong et al.'s survey [46]. In particular spectrum-based fault localization offers superior performance compared to static and dynamic program analysis applied to debugging. The open research question that is, whether spectrum-based fault localization can be efficiently used for software configuration diagnosis as well.
- Study and exploit the trade-off between the type of data from users required for diagnosis (as well as the effort of obtaining this data, e.g., via instrumentation) and the achieved accuracy. The research goals that would go into this direction include:
  - For each type of diagnosis data (from static analysis to diagnosis data dynamically created from instrumentation and also

for combinations) understand and quantify the degree of likely penalties (e.g., in terms of accuracy) of using only this data for diagnosis. Specifically, characterize error types which can be or *cannot be* diagnosed for each type of diagnosis data (when using state-of-the art debugging approaches).

– For each "class" of diagnosis data, attempt to improve the corresponding state-of-the art diagnosis methods in terms of types of errors they are able to debug. This can be done e.g., by an in-depth analysis why they fail for some error types and by providing substrates/replacements for the missing diagnosis data.

**Diagnosing of cross-stack configuration errors** In the case of cross-stack applications, there is not so much work available. Important open research challenges include:

- Exploit work on consistency checking to detect potential inconsistencies between different stack layers.
- Leverage existing work on extraction of rules and constraints to model dependencies *between* layers. Then use the techniques for discovery and fixing of constraint violations to diagnose (and possibly repair) cross-stack configuration errors.
- As a further application of extracted rules, configurator-like tools (as used for configuring operating systems) could be used for safe configuration of cross-stack systems.
- Create models of expected behavior (given a current global configuration) of each layer from the perspective of each layer. Divergences in the behavior might indicate potential configuration inconsistencies or errors. For example, given the current configuration of a database-layer (specifying $n_1$ database connections), also the PHP-layer should allow $n_1$ database connections. However, if the expected behavior of PHP-layer, based on its own configuration, allows only $n_2 < n_1$ database-connections, then an inconsistency between these two behavioral models is indicated.

It is worth noting that it is quite important which dependencies or interaction between layers can be observed or recorded. Moreover, in the context of these challenges the application of model-based approaches for diagnosing (configuration) knowledge-base, e.g., [8, 45, 9, 34], might be worth being considered.

**Testing-related challenges and goals** In case of testing, we are interested in detecting faults caused by configuration settings. There the motivation is to improve testing approaches specifically for detecting faults in system configurations ideally during software development. To clarify the meaning of "software testing" in context of configuration (errors) we should consider that an application failure in this context does not necessarily imply that there is a defect in code (as in traditional testing). Such a failure rather indicates that:

- There is a mismatch between the state of the application environment (operating system, file system, hardware, location of input data, libraries, network properties, remote components, etc.) and the configuration settings. This implies that a test for this type of error must take into consideration the environment.
- There is an inconsistency between configuration values, either within a single layer or between layers in a multi-layer application. The corresponding tests might be independent of the application environment, but are probably more comprehensive if this is also taken into account.

Consequently, this discussion gives rise to the following goals:

- Attempt automated test generation that considers the state of the application environment and the configuration settings (maybe implicitly). Such tests would adapt to environment changes and target only the above-mentioned mismatch between environment and configuration. In order to avoid confusion with the meaning of traditional testing, we might call this "configuration verification" step instead of testing.
- Generate tests that verify only the *consistency of configurations between layers* of a multi-stack system. In this case a test failure should indicate only an inconsistency, not a lack of adaptation to the production environment. For example, a test could only verify the consistency of configurations across layers, not execute the whole application.
- Generate tests which verify the correctness of application's behavior independently of the configuration settings. For example, an application should produce the same behavior independently of the exact path to input/output/libraries, number of used threads (in some range), used compiler (or its flags) etc.
- Generate tests that improve the outcome of fault localization. There it would be necessary to identify those tests that can distinguish different computed root causes (see e.g., [21]).

## 4 Threats to Validity

Several threats to validity of this paper exist. The main one is the risk of omitting important contributions to this field. To mitigate this risk, we have created lists of relevant works using several processes described below. We then merged and pruned the results according to the rank of the publishing venue and originality (i.e. works proposing a novel or distinctive approach were included even if published in a workshop). In the first literature collection process, we searched for publications containing the word "configuration" that were published in selected high-quality venues (ICSE, ASE, ISSTA, FSE, ISSRE, ICSME, ICPC, IEEE Trans. Software Eng., and some others) in the last five years; for each found publication, we verified via abstract whether a publication indeed targets configuration error (diagnosis). In the second process, we read the related work sections of the previously identified works, and created a list of papers discussed there, which are of relevance (here, also less prestigious venues were considered). Finally, we screened the survey [51] for checking that no important contribution was omitted.

Another threat to validity is the possibility to misinterpret any of the discussed papers (e.g. due to different understanding of terms), and state here inaccurate claims. To reduce this risk, we have studied each described contribution in a depth sufficient to avoid a misinterpretation. Besides of this, information from related work section to verify our interpretation was used where available.

## 5 Conclusion

In this paper, we presented a survey on methods and techniques used for detecting, localizing, and correcting faults in the context of software configurations. We distinguished the different cases of software

configuration diagnosis for single-layer and cross-stack applications as well as methods used in case of available configuration knowledge and further aspects. From the survey we were able to identify some still open challenges and research questions including distinguishing different variants of potential root causes, the lack of repositories of application-cases for validating and comparing research results as well as the need for new fault localization and testing methods.

The motivation for this paper is to provide a solid basis for future research in this area and to identify some important challenges in software configuration diagnosis worth being tackled. We also indicated some relationships with work on diagnosis of configuration knowledge bases and other approaches of software debugging that might stimulate this field. Because of the growing interest in providing programs comprising a stack of other programs that themselves can be configured, we see a growing need for research in this area.

## REFERENCES


[1] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J. C. van Gemund, 'A practical evaluation of spectrum-based fault localization', *Journal of Systems and Software*, **82**(11), 1780–1792, (2009).

[2] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund, 'Spectrum-based multiple fault localization', in *ASE 2009, 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, November 16-20, 2009*, pp. 88–99. IEEE Computer Society, (2009).

[3] Mona Attariyan and Jason Flinn, 'Automating Configuration Troubleshooting with Dynamic Information Flow Analysis', in *9th USENIX Conference on Operating Systems Design and Implementation*, pp. 1–11, Vancouver, BC, Canada, (2010). USENIX Association.

[4] Farnaz Behrang, Myra B. Cohen, and Alessandro Orso, 'Users Beware: Preference Inconsistencies Ahead', in *2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pp. 295–306, New York, NY, USA, (2015). ACM.

[5] Jon Brodkin. Why Gmail Went Down: Google Misconfigured Load Balancing Servers. `https://goo.gl/Hdga7H`. Accessed: 5 June 2018.

[6] Z. Dong, A. Andrzejak, D. Lo, and D. Costa, 'ORPLocator: Identifying Read Points of Configuration Options via Static Analysis', in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 185–195, (October 2016).

[7] Z. Dong, A. Andrzejak, and K. Shao, 'Practical and accurate pinpointing of configuration errors using static analysis', in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 171–180, (September 2015).

[8] A Felfernig, G Friedrich, D Jannach, and M Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', *Artificial Intelligence*, **152**(2), 213–234, (2004).

[9] A. Felfernig, M. Schubert, and C. Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **26**(1), 53–62, (2 2012).

[10] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal, 'Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis', in *32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pp. 497–508, Piscataway, NJ, USA, (2017). IEEE Press.

[11] Dongpu Jin, Myra B. Cohen, Xiao Qu, and Brian Robinson, 'PrefFinder: Getting the Right Preference in Configurable Software Systems', in *29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pp. 151–162, New York, NY, USA, (2014). ACM.

[12] Dongpu Jin, Xiao Qu, Myra B. Cohen, and Brian Robinson, 'Configurations Everywhere: Implications for Testing and Debugging in Practice', in *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pp. 215–224, New York, NY, USA, (2014). ACM.

[13] Sunint Kaur Khalsa and Yvan Labiche, 'An orchestrated survey of available algorithms and tools for combinatorial testing', in *25th International Symposium on Software Reliability Engineering*, pp. 323–334, (2015).

[14] Bogdan Korel and Janusz Laski, 'Dynamic Program Slicing', *Information Processing Letters*, **29**, 155–163, (1988).

[15] D. R. Kuhn, R. N. Kacker, and Y. Lei, 'Combinatorial testing', in *Encyclopedia of Software Engineering*, ed., Phillip A. Laplante, Taylor & Francis, (2012).

[16] D. Richard Kuhn, Renee Bryce, Feng Duan, Laleh Sh. Ghandehari, Yu Lei, and Raghu N. Kacker, 'Combinatorial testing: Theory and practice', in *Advances in Computers*, volume 99, 1–66, Elsevier, (2015).

[17] Max Lillack, Christian Kästner, and Eric Bodden, 'Tracking Load-time Configuration Options', in *29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pp. 445–456, New York, NY, USA, (2014). ACM.

[18] Dusica Marijan, 'Improving Configurable Software Testing with Statistical Test Selection', in *International Workshop on Formal Methods for Analysis of Business Systems*, ForMABS 2016, pp. 5–8, New York, NY, USA, (2016). ACM.

[19] J. Meinicke, C. P. Wong, C. Kästner, T. Thüm, and G. Saake, 'On essential configuration complexity: Measuring interactions in highly-configurable systems', in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 483–494, (September 2016).

[20] James Mickens, Martin Szummer, and Dushyanth Narayanan, 'Snitch: Interactive Decision Trees for Troubleshooting Misconfigurations', in *2Nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, pp. 8:1–8:6, Cambridge, MA, (2007). USENIX Association.

[21] Nica Mihai, Nica Simona, and Wotawa Franz, 'On the use of mutations and testing for debugging', *Software: Practice and Experience*, **43**(9), 1121–1142, (2013).

[22] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki, 'Where Do Configuration Constraints Stem From? An Extraction Approach and an Empirical Study', *IEEE Transactions on Software Engineering*, **41**(8), 820–841, (August 2015).

[23] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki, 'Mining Configuration Constraints: Static Analyses and Empirical Results', in *36th International Conference on Software Engineering*, ICSE 2014, pp. 140–151, New York, NY, USA, (2014). ACM.

[24] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel, 'Using Bad Learners to Find Good Configurations', in *2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pp. 257–267, New York, NY, USA, (2017). ACM.

[25] ThanhVu Nguyen, Ugur Koc, Javran Cheng, Jeffrey S. Foster, and Adam A. Porter, 'iGen: Dynamic Interaction Inference for Configurable Software', in *2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pp. 655–665, New York, NY, USA, (2016). ACM.

[26] Changhai Nie and Hareton Leung, 'A survey of combinatorial testing', *ACM Computing Surveys*, **43**(2), (January 2011).

[27] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund, 'Finding Near-optimal Configurations in Product Lines by Random Sampling', in *2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pp. 61–71, New York, NY, USA, (2017). ACM.

[28] Rahul Potharaju, Joseph Chan, Luhui Hu, Cristina Nita-Rotaru, Mingshi Wang, Liyuan Zhang, and Navendu Jain, 'ConfSeer: Leveraging Customer Support Knowledge Bases for Automated Misconfiguration Detection', *Proc. VLDB Endow.*, **8**(12), 1828–1839, (August 2015).

[29] Ariel Rabkin and Randy Katz, 'Precomputing Possible Configuration Error Diagnoses', in *2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pp. 193–202, Washington, DC, USA, (2011). IEEE Computer Society.

[30] Ariel Rabkin and Randy Katz, 'Static Extraction of Program Configuration Options', in *33rd International Conference on Software Engineering*, ICSE '11, pp. 131–140, New York, NY, USA, (2011). ACM.

[31] Vinod Ramachandran, Manish Gupta, Manish Sethi, and Soudip Roy Chowdhury, Determining Configuration Parameter Dependencies via Analysis of Configuration Data from Multi-tiered Enterprise Applications', in *6th International Conference on Autonomic Computing*, ICAC '09, pp. 169–178, New York, NY, USA, (2009). ACM.

[32] M. Sayagh and B. Adams, 'Multi-layer software configuration: Empirical study on wordpress', in *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 31–40, (September 2015).

[33] Mohammed Sayagh, Noureddine Kerzazi, and Bram Adams, 'On Cross-stack Configuration Errors', in *39th International Conference on Software Engineering*, ICSE '17, pp. 255–265, Piscataway, NJ, USA, (2017). IEEE Press.

[34] Kostyantyn Shchekotykhin, Gerhard Friedrich, Patrick Rodler, and Philipp Fleiss, 'Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation', in *ECAI '14*, pp. 813–818, (2014).

[35] K. Shi, 'Combining Evolutionary Algorithms with Constraint Solving for Configuration Optimization', in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 665–669, (September 2017).

[36] S. Souto, M. D'Amorim, and R. Gheyi, 'Balancing Soundness and Efficiency for Practical Testing of Configurable Systems', in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 632–642, (May 2017).

[37] Ya-Yunn Su, Mona Attariyan, and Jason Flinn, 'AutoBash: Improving Configuration Management with Operating System Causality Analysis', in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, pp. 237–250, Stevenson, Washington, USA, (2007). ACM.

[38] T. Uchiumi, S. Kikuchi, and Y. Matsumoto, 'Misconfiguration detection for cloud datacenters using decision tree analysis', in *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*, pp. 1–4, (September 2012).

[39] Bo Wang, Leonardo Passos, Yingfei Xiong, Krzysztof Czarnecki, Haiyan Zhao, and Wei Zhang, 'SmartFixer: Fixing Software Configurations Based on Dynamic Priorities', in *17th International Software Product Line Conference*, SPLC '13, pp. 82–90, New York, NY, USA, (2013). ACM.

[40] Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi-min Wang, 'Automatic Misconfiguration Troubleshooting with PeerPressure', in *In OSDI*, pp. 245–258, (2004).

[41] Mengliao Wang, Xiaoyu Shi, and K. Wong, 'Capturing Expert Knowledge for Automated Configuration Fault Diagnosis', in *2011 IEEE 19th International Conference on Program Comprehension (ICPC)*, pp. 205–208, (June 2011).

[42] Yi-min Wang, Chad Verbowski, John Dunagan, Yu Chen, Helen J. Wang, and Chun Yuan, 'STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support', in *In Usenix LISA*, pp. 159–172, (2003).

[43] Mark Weiser, 'Program slicing', *IEEE Transactions on Software Engineering*, **10**(4), 352–357, (July 1984).

[44] Andrew Whitaker, Richard S. Cox, and Steven D. Gribble, 'Configuration Debugging As Search: Finding the Needle in the Haystack', in *6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pp. 6–6, San Francisco, CA, (2004). USENIX Association.

[45] Jules White, David Benavides, Douglas C. Schmidt, Pablo Trinidad, Brian Dougherty, and Antonio Ruiz Cortés, 'Automated diagnosis of feature model configurations', *Journal of Systems and Software*, **83**(7), 1094–1107, (2010).

[46] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa, 'A survey on software fault localization', *IEEE Trans. Software Eng.*, **42**(8), 707–740, (2016).

[47] Y. Xiong, H. Zhang, A. Hubaux, S. She, J. Wang, and K. Czarnecki, 'Range Fixes: Interactive Error Resolution for Software Configuration', *IEEE Transactions on Software Engineering*, **41**(6), 603–619, (June 2015).

[48] Yingfei Xiong, Arnaud Hubaux, Steven She, and Krzysztof Czarnecki, 'Generating Range Fixes for Software Configuration', in *34th International Conference on Software Engineering*, ICSE '12, pp. 58–68, Piscataway, NJ, USA, (2012). IEEE Press.

[49] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker, 'Hey, You Have Given Me Too Many Knobs!: Understanding and Dealing with Over-designed Configuration in System Software', in *2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pp. 307–319, New York, NY, USA, (2015). ACM.

[50] Tianyin Xu, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy, 'Do Not Blame Users for Misconfigurations', in *Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 244–259, Farminton, Pennsylvania, (2013). ACM.

[51] Tianyin Xu and Yuanyuan Zhou, 'Systems Approaches to Tackling Configuration Errors: A Survey', *ACM Comput. Surv.*, **47**(4), 70:1–70:41, (July 2015).

[52] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram, and Shankar Pasupathy, 'An Empirical Study on Configuration Errors in Commercial and Open Source Systems', in *Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pp. 159–172, New York, NY, USA, (2011). ACM.

[53] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy, 'SherLog: Error Diagnosis by Connecting Clues from Run-time Logs', in *Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pp. 143–154, New York, NY, USA, (2010). ACM.

[54] Ding Yuan, Yinglian Xie, Rina Panigrahy, Junfeng Yang, Chad Verbowski, and Arunvijay Kumar, 'Context-based Online Configuration-error Detection', in *2011 USENIX Conference on USENIX Annual Technical Conference*, pp. 28–28, Portland, OR, (2011). USENIX Association.

[55] Jiaqi Zhang, Lakshminarayanan Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, Tianyin Xu, and Yuanyuan Zhou, 'EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection', in *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 687–700, Salt Lake City, Utah, USA, (2014). ACM.

[56] Sai Zhang, 'ConfDiagnoser: An Automated Configuration Error Diagnosis Tool for Java Software', in *2013 International Conference on Software Engineering*, ICSE '13, pp. 1438–1440, Piscataway, NJ, USA, (2013). IEEE Press.

[57] Sai Zhang and Michael D. Ernst, 'Automated diagnosis of software configuration errors', in *ICSE'13, 34th International Conference on Software Engineering*, San Francisco, CA, USA, (May 2013).

[58] Sai Zhang and Michael D. Ernst, 'Which Configuration Option Should I Change?', in *36th International Conference on Software Engineering*, ICSE 2014, pp. 152–163, New York, NY, USA, (2014). ACM.

[59] Sai Zhang and Michael D. Ernst, 'Proactive Detection of Inadequate Diagnostic Messages for Software Configuration Errors', in *Int. Symp. on Software Testing and Analysis (ISSTA)*, pp. 12–23, NY, USA, (2015). ACM.

# Liquid Democracy in Group-based Configuration

**Muesluem Atas** and **Thi Ngoc Trang Tran** and **Ralph Samer**
and **Alexander Felfernig** and **Martin Stettinger**
**Institute of software technology, Graz University of Technology, Graz, Austria,**
**email: {muesluem.atas, ttrang, rsamer, alexander.felfernig, mstettinger}@ist.tugraz.at**

**Davide Fucci**
**University of Hamburg, Hamburg, Germany**
**email: fucci@informatik.uni-hamburg.de**

**Abstract.** Group-based configuration systems support scenarios where a group of users configures a product/service. In those group-based configuration scenarios where the knowledge of some group members regarding items is insufficient, an advice of experts is necessary in order to help members to evaluate products or services. This paper introduces a novel approach which takes advantage of the concept of *liquid democracy* that allows the delegation of group member votes to experts. Concerning the application of *liquid democracy*, we propose a new approach based on *Multi-attribute Utility theory (MAUT)-based evaluation* used to calculate the utility of configurable items. Compared to the traditional approach, the proposed MAUT-based evaluation focuses on the role of experts by assigning higher weights to them. Additionally, the respective expertise level of the experts is taken into account. Consequently, the main contribution of this paper consists in the improvement of group-based configuration by taking *liquid democracy* aspects into consideration.

## 1 Introduction

Configuration [5, 12] is an important application area of Artificial Intelligence that enables users to configure *complex items* described by many dimensions (attributes). Typical examples of such items include *release plans* [10], *tourism packages* [13], *furnitures* [6], and *financial services* [7, 11]. While most existing configuration systems focus on the support of single users, there also exist scenarios where items can be jointly configured by groups of users, e.g., requirements engineering scenarios where a group of stakeholders configures software release plans. In such scenarios, *group-based configuration systems* have been recognized as being useful tools that help to identify configurations which satisfy preferences of all group members [4]. When interacting with group-based configuration systems, each group member explicitly articulates his/her preference with respect to different item dimensions. Preferences declared by group members are then checked for consistency. As soon as all user preferences are consistent with each other as well as with the knowledge base, the constraint solver will be able to find items that satisfy the preferences of all group members. After this, utility values for each item can be calculated, for example, on the basis of *Multi-attribute Utility theory (MAUT)* [3]. Such an approach takes into account the preferences of group members with respect to the dimensions of items and the importance of dimensions from the users' point of view. The item achieving the highest utility value will then be recommended to the group.

In the context of group-based configuration, sometimes, some group members may be *unable* to evaluate the dimensions of a given set of items due to a knowledge gap. Hence, in order to precisely evaluate items, group members have to invest much effort in order to collect necessary information as well as to analyze items [14]. In such a situation, group members could ask for advice from people who are experts in the item domain of interest. The consultation of experts helps to precisely identify evaluations of items and thereby further facilitates the entire configuration process. The preference configuration of group members in this context can be interpreted and considered as a *liquid democracy* paradigm which provides an alternative decision making model to make better use of collective intelligence [14]. The *liquid democracy* concept empowers group members to either play an *active role* (i.e., *active users* who directly vote items) or a *passive role* (i.e., *passive users* who delegate their rating power to experts) in the voting process [2].

Recently, a variety of studies regarding liquid democracy have been conducted for the purpose of making better use of the so-called "*wisdom of the crowds*" [14]. For instance, Boldi et al. [2] propose a Facebook application that enables each user to select one of his/her friends as the expert of a music genre. The expert then helps him/her to select some pieces of music. Johann et al. [8] introduce the applicability of liquid democracy and e-democracy concepts to address challenges of a massive and continuous user participation in the context of requirements engineering. Zhang et al. [14] propose an efficient *statement voting* scheme that unifies two basic stages of liquid democracy, i.e., *delegation* and *voting*. During the voting/delegating phase, each voter can either vote for candidate(s) or delegate his/her voting power to another voter. Each voter is assigned with a *temporal ID* which is encrypted and distributed in such a way that guarantees the anonymity of the delegation/voting process. Although up to now, there exist many studies with regard to liquid democracy, to some extent it is still unclear how liquid democracy can be applied in the context of group-based configuration. Two emerging questions are: (i) "*How does the system recommend experts to a user who has not enough knowledge about items?*" and (ii) "*How to calculate the utility on the basis of emphasizing the importance of experts who were chosen by stakeholders?*". To the best of our knowledge, there does not exist any research

which provides an in-depth view of the correct application of liquid democracy in group-based configuration. In this paper, we present an insight of the application of liquid democracy in group-based configuration and propose a novel approach of a MAUT-based evaluation that takes preferences of group members/experts into account and thereby assigns a higher importance to the experts.

The remainder of the paper is organized as follows. In *Section 2*, we describe a group-based configuration scenario in requirements engineering which is used as a working example throughout the paper. In *Section 3*, we discuss how liquid democracy can be applied to group-based configuration in order to transfer the rating power from group members to experts. *Section 4* presents a new approach of MAUT-based evaluation to calculate the utility value of a requirement. *Section 5* discusses how requirements can be assigned to releases based on their utility value, their effort estimation, existing dependencies between requirements, and the capacity of releases. Finally, Section 6 draws a brief conclusion and provides some ideas for future work.

## 2 Working example

For demonstration purposes, we introduce a group configuration scenario occurring in a small requirements engineering example project where we configure a release plan. In this context, we define a set of requirements ($R_1$, $R_2$, $R_3$, and $R_4$) for developing a *sport watch*. These requirements are defined by a group of engineers with longstanding experience and practical knowledge in requirements engineering. Each requirement is described by an *id*, a *title*, and a textual *description* (see Table 1).

| Id | Title | Description |
|---|---|---|
| $R_1$ | Evaluation Software | To evaluate the collected training data, an evaluation software is required. The evaluation software requires the connection and the access to the clock's internal memory. The evaluation should contain measured information regarding the distance, the height, the average heart rate, and the calorie consumption. |
| $R_2$ | Data-Storage Function | To evaluate the measured data, a storage service is required. The internal memory is used for saving the measured information, such as the distance, the height, the average heart rate, and the calorie consumption. The stored data will be used by the evaluation software. |
| $R_3$ | GPS | To identify the position, a GPS sensor is used. Based on the measured position and time information, the speed and the distance can be measured. |
| $R_4$ | Display lighting | The sport watch needs a display lighting to be operated at dusk. |

**Table 1.** Example requirements for the development of a *sport watch*. Each requirement is described by an *id*, a *title*, and a textual *description*.

In this example, we assume a situation where a group of five stakeholders (i.e., users) reads requirements, evaluates them regarding different dimensions of requirements, and assigns them to different releases (i.e., release planning configuration). We defined two different releases which are shown in Table 2.

Given the sets of requirements and releases, we assume that each stakeholder evaluated requirements with regard to the following dimensions: *risk, effort*, and *profit*. The dimension of *risk* indicates "the estimated risk for developing a requirement", *effort* represents "the estimated total work done for developing a requirement", and *profit* corresponds to "the estimated profit of a requirement". These

| Releases | Capacity (in hours) | Start date | End date |
|---|---|---|---|
| *Release 1* | 260 | 2020-05-01 | 2020-07-01 |
| *Release 2* | 260 | 2020-07-15 | 2020-09-15 |

**Table 2.** Defined releases for the development of a *sport watch*. Each release is described by the *start date*, the *end date,* and the *capacity*. The *capacity* indicates the planned effort of a release in hours.

dimensions are evaluated using ratings. Thereby, the ratings can lie in the range between 1 and 5, where an evaluation of 5 indicates a requirement with *low risk, high profit,* and *low effort* and an evaluation of 1 represents a requirement with *high risk, low profit,* and *high effort*.

The evaluation of stakeholders with regard to different dimensions of requirements is shown in Table 3. In this table, some group members did not sufficiently evaluate dimensions of a requirement (i.e., some dimensions were not evaluated by stakeholders). For instance, the first stakeholder ($S_1$) did not evaluate the *profit* of the requirement $R_2$. In addition, there also exist some stakeholders who did not evaluate any dimension of a requirement. For instance, the fourth stakeholder ($S_4$) did not evaluate any dimension of the requirement $R_1$. A possible reason for the existence of such missing evaluations can be lacking expertise or knowledge of some stakeholders regarding the meaning of some requirements. In this scenario, those stakeholders who do not know much about the content of some requirements have to invest a lot of effort in order to acquire enough necessary knowledge and to analyze the requirements. This triggers a high cost of the requirement evaluation process. Alternatively, the stakeholder could ask for the advice of some experts to provide more precise evaluations with regard to dimensions of requirements. In other words, the stakeholder directly passes his/her evaluation power to experts by using liquid democracy (see Section 3). In the context of requirements engineering, the expert can be a requirement engineer who has longstanding experiences and practical knowledge of requirements. The consultation of experts helps to precisely evaluate the dimensions of items.

Alternatively, in some cases empty evaluations could be trigged by the reason that the stakeholder does not want to evaluate some dimensions of a requirement. Moreover, he/she also does not want to delegate the rating power to anyone else. In this scenario, group-based configuration systems will automatically check the quantity of complete evaluations of requirements and the configuration phase is only complete if this quantity is high enough. In this example, we assume that the quantity of complete evaluations should be *equal to or greater than 80%* of the total number of all evaluations. In other words, the configuration phase will not be finished until the quantity of available evaluations reaches 80%.

In addition to that, when evaluating a requirement, stakeholders can assign different *weights* to dimensions. Thereby, the weight is referred to as the importance of a dimension, i.e., the higher the importance of a dimension, the higher the weight. Different stakeholders could assign different weights to the same dimension. For instance, one stakeholder (e.g., developer) assumes that the *effort* of a requirement is the most important dimension, whereas another stakeholder (e.g., project manager) evaluates the *profit* to be the most important dimension of a requirement. In order to limit the scope of this paper, some simplifications have to be made. For the sake of simplicity, we assume that the importance (i.e., weight) of all dimensions for all stakeholders is equal and all dimensions have a weight of 1 from the

| Stakeholders | Requirement 1 ($R_1$) | | | Requirement 2 ($R_2$) | | | Requirement 3 ($R_3$) | | | Requirement 4 ($R_4$) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Risk | Profit | Effort | Risk | Profit | Effort | Risk | Profit | Effort | Risk | Profit | Effort |
| $S_1$ | 5 | 3 | 4 | 2 | - | 4 | 4 | 4 | 4 | 2 | - | - |
| $S_2$ | 3 | 3 | - | 2 | - | 3 | 2 | 5 | - | 2 | 5 | 4 |
| $S_3$ | 3 | 4 | 3 | 5 | - | 4 | 2 | 3 | 3 | 4 | 2 | - |
| $S_4$ | - | - | - | 4 | - | 2 | 2 | 4 | 4 | 1 | 3 | - |
| $S_5$ | 3 | 3 | 4 | - | - | 4 | 2 | - | 4 | 4 | 3 | 4 |

**Table 3.** Evaluations of stakeholders with regard to the defined requirements in Table 1. Each requirement is represented by the three following dimensions: *risk, profit,* and *effort*. Each evaluation is in the range of 1 to 5, where the evaluation of 5 indicates a requirement with *low risk, high profit,* and *low effort* and the evaluation of 1 represents a requirement with *high risk, low profit,* and *high effort*. Evaluations which were not provided by stakeholders are represented as a dash symbol ("-").

stakeholders' point of view (i.e., $\forall$ s $\in$ stakeholders, *weight*(*s, risk*) = *weight*(*s, profit*) = *weight*(*s, effort*) = 1).

## 3 Application of Liquid Democracy

Liquid democracy is a hybrid voting model of participative democracy which combines *direct* and *representative democracy* approaches in order to empower electors [1, 9]. While *direct democracy* allows electors to directly vote for an item, *representative democracy* enables electors to select representatives (or experts) and empower them to vote for items. One of the major issues of direct democracy is the insufficient knowledge of the voter about some items. As a consequence, these voter may provide unprecise evaluations or may even not be able to assess them in a reasonable way. In sharp contrast to direct democracy, representative democracy allows a stakeholder to elect an expert who plays the role of a representative to vote for items. However, representative democracy is also known to show a weakness in terms of *representativeness*. In particular, this is true for scenarios where many voters delegate their voting power to only one expert. That means, the expert's opinion usually represents the idea of many voters and hence, it triggers a situation in which the evaluation of the expert partly reflects the opinion of a stakeholder. In this context, liquid democracy has been recognized as a mixed approach which takes advantage of the strength of direct and representative democracy. Liquid democracy enables voters to either directly vote items or delegate their voting rights to an expert. Consequently, this key benefit of liquid democracy serves as main motivation to apply this voting model.

In this paper, we use a liquid democracy approach in order to complete evaluations of dimensions which were not evaluated by stakeholders. As shown in Table 3, we can observe that stakeholders did not evaluate all dimensions of requirements. In this example, we assume that stakeholders $S_2$, $S_4$, and $S_5$ need experts to complete their evaluations. Expert selection can be done by one of two approaches. The first approach is to select *only one expert* for the three aforementioned stakeholders. The second approach is to allow each group member to select his/her own expert. In our example, we choose the second approach where each stakeholder chooses different experts for different requirements. For instance, regarding the requirement which is related to the user interface, the stakeholder can choose an expert who has experiences in user interface design. For the data storage-related requirement, the stakeholder can choose an expert who has knowledge of data management. In our approach, the expert selection process is done automatically by the recommender system. That means, experts on a specific topic are automatically identified and recommended to the stakeholder. Alternatively, each stakeholder is allowed to select experts who are not included in the recommended list. In our approach, the recommender system suggests experts based on the *expertise level*. In the context of requirements engineering, the expertise level of an expert can be calculated based on the following criteria: *working experience*, *skills*, *number of contributions* in requirements engineering projects, and *number of delegations* received in the requirements engineering domain. The expertise level is in range of 1 to 5. The expertise level of 5 indicates excellent topic-related knowledge, whereas the expertise level of 1 represents limited knowledge.

In this paper, we exemplify an expert recommendation process with 5 experts in the requirements engineering domain. Table 4 shows a recommended list of experts ranked in a descending order of the expertise level. In addition, stakeholders who want to delegate evaluations to other experts can select different experts for different requirements. As shown in Table 1, the development of the defined requirements requires deep knowledge with regard to different areas. Therefore, selecting an appropriate expert for each requirement helps to increase the overall quality of requirements engineering. The expert selection for stakeholders $S_2$, $S_4$, and $S_5$ are depicted in Table 5. In this table, we can observe that stakeholders select different experts for different requirements. For instance, stakeholder $S_2$ requires experts' evaluations regarding the dimensions of requirements $R_1$, $R_2$, and $R_3$. The stakeholder $S_2$ chooses $Expert_3$ for $R_1$, $Expert_2$ for $R_2$, and $Expert_4$ for $R_3$. Furthermore, we can observe that the stakeholder $S_2$ does not need any expert for $R_4$ and this is represented by a dash symbol ("-") in Table 5.

| Experts | Expertise-Level (*sport watch* domain) |
|---|---|
| $Expert_2$ | 4.5 |
| $Expert_5$ | 3.75 |
| $Expert_4$ | 3.15 |
| $Expert_1$ | 2.25 |
| $Expert_3$ | 2.05 |

**Table 4.** The expertise level in the *sport watch* domain. The expertise level is in the range of 1 to 5, whereby 1 indicates limited knowledge and 5 indicates excellent knowledge.

| Stakeholders | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| $S_2$ | $Expert_3$ | $Expert_2$ | $Expert_4$ | - |
| $S_4$ | $Expert_2$ | $Expert_2$ | - | $Expert_2$ |
| $S_5$ | - | $Expert_5$ | $Expert_2$ | - |

**Table 5.** Experts chosen by stakeholders regarding different requirements. The dash symbol "-" represents a situation in which a stakeholder does not need any advice of an expert.

After the selection of experts with respect to each requirement, these experts evaluate the remaining requirement dimensions which

| Stakeholders | Requirement 1 ($R_1$) | | | Requirement 2 ($R_2$) | | | Requirement 3 ($R_3$) | | | Requirement 4 ($R_4$) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Risk | Profit | Effort | Risk | Profit | Effort | Risk | Profit | Effort | Risk | Profit | Effort |
| $S_1$ | 5 | 3 | 4 | 2 | - | 4 | 4 | 4 | 4 | 2 | - | - |
| $S_2$ | 3 | 3 | **3** | 2 | **3** | 3 | 2 | 5 | **3** | 2 | 5 | 4 |
| $S_3$ | 3 | 4 | 3 | 5 | - | 4 | 2 | 3 | 3 | 4 | 2 | - |
| $S_4$ | **2** | **3** | **3** | 4 | **4** | 2 | 2 | 4 | 4 | 1 | 3 | **5** |
| $S_5$ | 3 | 3 | 4 | **3** | **2** | 4 | 2 | **4** | 4 | 4 | 3 | 4 |

**Table 6.** Evaluations of stakeholders with regard to the defined requirements in Table 1. Each requirement is represented by the three following properties: *risk, profit,* and *effort*. The evaluation is in the range of 1 to 5, where the evaluation of 5 indicates a requirement with *low risk, high profit,* and *low effort* and the evaluation of 1 represents a requirement with *high risk, low profit,* and *high effort*. Evaluations which were not provided by stakeholders and experts are represented as a dash symbol ("-"). Evaluations provided by experts are represented in bold text.

were not evaluated by the stakeholders $S_2$, $S_4$, and $S_5$. The evaluations which were given by experts are shown (in bold) in Table 6. Next, the utility of each requirement has to be calculated and used as one of the important criteria to assign requirements to releases. The utility of each requirement is calculated based on *Multi-attribute Utility Theory* (*MAUT*) (see Section 4).

## 4 Application of Multi Attribute Utility Theory

As already mentioned before, configurable items are usually described by a set of dimensions. In this context, *Multi-attribute Utility Theory* (*MAUT*) [3] is applied. In this paper, we propose a new MAUT-based approach that calculates the utility of an item *i* according to the evaluations of stakeholders (*evaluation(s,d)*) with regard to dimensions $d$, the importance of these dimensions (*w(s,d)*) from the stakeholders' point of view, and the importance of stakeholders/experts (*w(s)*). The final result of the MAUT evaluation is then represented by the *weighted average* of all stakeholders' evaluations for the dimensions $d$. Formula 1 indicates that an experts' evaluation *evaluation(e,d)* for a dimension $d$ is used in cases where a stakeholders' voting is delegated. Otherwise, a stakeholders' own evaluation will be taken into account for the MAUT calculation. In our approach, compared to a stakeholder, an expert has a higher impact on the overall utility of an item, i.e., the weight of an expert is twice the weight of a stakeholder (see Formula 2). In addition, the expertise level *el(e)* of an expert *e* is also considered in the weight calculation. The total MAUT value (i.e., the utility value) of a requirement $R_i$ is then calculated by summing all dimension-specific MAUT values of the requirement $R_i$ (see Formula 3).

$$eval(s,d) = \begin{cases} evaluation(e,d) & \text{if} \textit{evaluation(s,d) delegated} \\ evaluation(s,d) & \textit{otherwise} \end{cases} \tag{1}$$

$$w(s) = \begin{cases} weight(s) * 2 + el(e) & \text{if } \textit{evaluation(s,d) delegated} \\ weight(s) & \textit{otherwise} \end{cases} \tag{2}$$

$$Utility(R_i) = \frac{\sum_{s \in stakeholders} \frac{\sum_{d \in dims} eval(s,d) * w(s,d) * w(s)}{\sum_{d \in dims} w(s,d) * w(s)}}{|stakeholders|} \tag{3}$$

An example of the utility calculation of a requirement is presented in Formula (4). In this example, for simplicity, we assume that all stakeholders assign the same weight (i.e., the weight of 1) for all dimensions of requirements ($\forall s \in stakeholders, \forall d \in dimensions$, w($s, d$)= 1). Additionally, we assume that each stakeholder has also the same importance ($\forall s \in stakeholders$ weight($s$)= 1).

$$\begin{aligned} Utility(R_2) &= \frac{\sum_{s \in stakeholders} \frac{\sum_{d \in dims} eval(s,d) * w(s,d) * w(s)}{\sum_{d \in dims} w(s,d) * w(s)}}{|stakeholders|} \\ &= \frac{1}{5}\Big(\frac{2*1+4*1}{1+1} + \frac{2*1+3*(1*2+4.5)+3*1}{1+(1*2+4.5)+1} \\ &+ \frac{5*1+4*1}{1+1} + \frac{4*1+4*(1*2+4.5)+2*1}{1+(1*2+4.5)+1} \\ &+ \frac{3*(1*2+3.75)+2*(1*2+3.75)+4*1}{(1*2+3.75)+(1*2+3.75)+1}\Big) \\ &= \frac{1}{5}\Big(\frac{6}{2} + \frac{24.5}{8.5} + \frac{9}{2} + \frac{32}{8.5} + \frac{32.75}{12.5}\Big) = 3.354 \end{aligned} \tag{4}$$

Similarly, MAUT values of other requirements are also calculated by using Formulae 1 - 3. The MAUT values of requirements $R_1, R_2, R_3$, and $R_4$ are the following: $MAUT(R_1) = 3.266$, $MAUT(R_2) = 3.354$, $MAUT(R_3) = 3.380$ and $MAUT(R_4) = 3.326$. After the calculation of requirement utilities, requirements will be assigned to defined releases (see Section 5).

## 5 Release Planning

In Section 4, we showed how the utility value of a requirement can be calculated based on *Multi-attribute Utility Theory* (*MAUT*). The higher the MAUT value, the sooner the requirement will be implemented. In the context of requirements engineering, making a recommendation of requirements is referred to as release planning, i.e., to clarify which requirement should be implemented in which release. In release planning, stakeholders have to estimate the effort investing for each requirement. In our working example, the *effort* is referred to as the invested time (in hour) to implement a requirement. The higher the evaluation of effort, the lower the invested time. We assume that an evaluation of 5 corresponds to an effort of 50 hours and an evaluation of 1 corresponds to an effort of 250 hours. In order to calculate the effort of a requirement, we first calculate the average of evaluations with regard to the effort of the requirement given by all stakeholders. After that, the effort (in hour) is calculated using the Formula 5, where $effort(R_i, s)$ is the evaluation of the stakeholder $s$ about the effort of the requirement $R_i$.

$$effort(R_i) = \Big(5 - \frac{\sum_{s \in stakeholders} effort(R_i, s)}{|stakeholders|} + 1\Big) * 50 \tag{5}$$

We exemplify the calculation of the effort of the requirement $R_1$ as shown in Formula 6. The effort values of other requirements are calculated in a similar way and presented in Table 7.

$$effort(R_1) = \left(5 - \frac{4+3+3+3+4}{5} + 1\right) * 50 = 130 \quad (6)$$

| Requirements | Average effort (effort in hours) | Assigned release |
|---|---|---|
| $R_1$ | 3.4 (130) | Release 2 |
| $R_2$ | 3.4 (130) | Release 1 |
| $R_3$ | 3.6 (120) | Release 1 |
| $R_4$ | 4.33 (83.33) | Release 2 |

**Table 7.** The assignment of requirements to releases based on the effort of requirements, dependencies between requirements, their utility values, and capacity of releases. The effort of each requirement in represented in the second column of the table.

In our example, release planning is done based on four criteria: the *utility* (MAUT) value, *effort* (measured in hours), the *dependency* between requirements, and the *capacity* of releases. Given the fact that requirement $R_3$ achieves the highest utility (i.e., $MAUT(R_3) = 3.380$) and its estimated time effort of 120 hours, $R_3$ turns out to be the best candidate to be assigned to *Release 1*. Furthermore, it is reasonable that requirement $R_2$ should follow $R_3$ and hence also be assigned to *Release 1*, as $R_2$ shows the second highest utility and there is still some remaining capacity left for *Release 1* to cover $R_2$ in this release (i.e., capacity(*Release 1*) = 260 hours). Next, requirement $R_4$ has to be assigned to a release. The assignment of requirement $R_4$ to the first release is not possible, because of the limited capacity ( remaining capacity (*Release 1*) = 10 hours and effort($R_4$) = 83.33 hours). Therefore, requirement $R_4$ is assigned to the second release. Finally, requirement $R_1$ is assigned to the second release. Based on the description of requirements shown in Table 1, we can observe that there is a dependency between $R_1$ and $R_2$ which is indicated as follows: "*The evaluation software requires the access to the clock's internal memory*". In the context of requirements engineering, this means that the first requirement $R_1$ (i.e, evaluation software) can not be implemented before $R_2$ (i.e., data storage function) has been completed. However, in this scenario, the identified dependency will not trigger any changes in the release planning since the release plan in Table 7 shows that requirement $R_2$ which is assigned to the first release (development period: from 2020-05-01 to 2020-07-01) will be implemented before all the requirements assigned to the second release (development period: from 2020-07-15 to 2020-09-15). With this final step, all requirements are assigned to releases and the requirements engineering process is complete.

## 6 Conclusion and Future Work

In this paper, we introduced utility analysis concepts which focus on *liquid democracy*. These concepts allow the manual delegation of a stakeholder's voting right to a domain expert. First, we described a scenario for the development of a sport watch which is used as a working example throughout this paper. Based on the working example, we applied liquid democracy in order to receive consultations from experts in situations where stakeholders do not have enough knowledge with regard to certain requirements. Afterwards, we proposed a novel approach of a MAUT-based evaluation which takes into account users' and experts' evaluations and assigns higher importance to expert consultations (i.e., evaluations). Finally, we proposed a group-based configuration for release planning where requirements were assigned to releases based on derived utility values, effort estimations, existing dependencies, and release capacities.

Within the scope of future work, we plan to integrate the proposed approach in a requirements engineering tool named INNOSENSR[1]. INNOSENSR is a modern innovative release planning tool which makes use of intelligent techniques in order to facilitate the complete requirements engineering process. In the current version of INNOSENSR, stakeholders have to evaluate requirements without getting any support from domain experts. In the future, we will integrate our approach into INNOSENSR in order to increase the requirements engineering quality.

## Acknowledgment

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OPENREQ (732463).

## REFERENCES

[1] Christian Blum and Christina Isabel Zuber, 'Liquid democracy : Potentials, problems, and perspectives', *The Journal of Political Philosophy*, **24**, 162–182, (2016).

[2] Paolo Boldi, Corrado Monti, Massimo Santini, and Sebastiano Vigna, 'Liquid FM: recommending music through viscous democracy', in *Proceedings of the 6th Italian Information Retrieval Workshop, Cagliari, Italy, May 25-26, 2015.*, (2015).

[3] J. S. Dyer, *Maut - Multi-attribute Utility Theory*, 265–292, Springer New York, New York, NY, 2005.

[4] Alexander Felfernig, M Atas, TNT Tran, and Martin Stettinger, 'Towards group-based configuration', in *International Workshop on Configuration 2016 (ConfWS16)*, pp. 69–72, (2016).

[5] Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edn., 2014.

[6] A. Haag, 'Sales configuration in business processes', *IEEE Intelligent Systems and their Applications*, **13**(4), 78–85, (Jul 1998).

[7] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich, *Recommender Systems: An Introduction*, Cambridge University Press, New York, NY, USA, 1st edn., 2010.

[8] Timo Johann and Walid Maalej, 'Democratic mass participation of users in requirements engineering?', in *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*, pp. 256–261, (2015).

[9] Anna Litvinenko, 'Social media and perspectives of liquid democracy on the example of political communication of pirate party in germany', in *The Proceedings of the 12th European Conference on e-Government in Barcelona*, pp. 403–408, (2012).

[10] Gerald Ninaus, Alexander Felfernig, Martin Stettinger, Stefan Reiterer, Gerhard Leitner, Leopold Weninger, and Walter Schanil, 'Intellireq: Intelligent techniques for software requirements engineering', in *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14, pp. 1161–1166, Amsterdam, The Netherlands, The Netherlands, (2014). IOS Press.

[11] Markus Stolze, Simon Field, and Pascal Kleijer, 'Combining configuration and evaluation mechanisms to support the selection of modular insurance products', in *Proceedings of the 8th European Conference on Information Systems, Trends in Information and Communication Systems for the 21st Century, ECIS 2000, Vienna, Austria, July 3-5, 2000*, pp. 858–865, (2000).

[12] Markus Stumptner, 'An overview of knowledge-based configuration', *Artificial Intelligence Community*, **10**(2), 111–125, (April 1997).

[13] TNT Tran, Müslüm Atas, Martin Stettinger, and Alexander Felfernig, 'An extension of choicla user interfaces for configurable products', *RSBDA'16*.

[14] Bingsheng Zhang and Hong-sheng Zhou, 'Brief announcement: Statement voting and liquid democracy', in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pp. 359–361, New York, NY, USA, (2017). ACM.

[1] `http://innosensr.com`

# Knowledge retrieval for configuring risks when answering calls to tenders or direct customer demands

**Ayachi Rania**[1,2] and **Guillon Delphine**[1,3] and **Vareilles Elise**[1] and **Marmier François**[1]

**Aldanondo Michel**[1] and **Coudert Thierry**[2] and **Geneste Laurent**[3]

**Abstract.** This short article provides the first ideas and results about the configuration of risks when answering tenders or direct customer demands. Indeed, when an offer is defined, it becomes more and more important to analyze possibilities of risks occurrence and their consequences. Most of the time, this analysis is conducted manually thanks to a risk expert. In this paper, we propose to assist the expert with a risk configuration tool that relies on a knowledge base and that allows to define and evaluate: (i) the risk probability, (ii) the main risk impacts and (iii) the interests of various corrective and preventive actions to mitigate it. We first detail the problem. Then we propose a generic model of risks for calls for tenders. Then we describe some knowledge retrieval queries that support the configuration of risk characteristics. As preliminary studies, we will not be able to discuss hard theoretical results but should be able to show a nice a demo of a first software prototype.

*Keywords:* Customer/supplier relation, offer elaboration, risk configuration, knowledge based system, knowledge model, case base reasoning

## 1 INTRODUCTION

This short article deals with offer elaboration when answering call for tenders or direct customer demands. The offer concerns physical product or mechanical systems, called indistinctly in the rest of the paper 'systems'. The customer/supplier relation is assumed to be in a B2B context and in a "light" Engineer To Order situation (ETO) [1]. By light ETO we mean that more than 75% of the systems are configured to order (CTO), either Assembly or Make To Order (ATO or MTO); the 25% left are engineer to order (ETO). Globally, such systems are mainly standard but allow some customer specific options that are non-standard, also called ETO options [2]. These ETO options are a strong point for the supplier's competitiveness.

During the offer elaboration, as there is no guarantee that the customer accepts the offer, we assume that the supplier doesn't study in detail: (i) the design of every ETO option, (ii) their integration with the standard solution and (iii) their production process. The supplier configures in detail the CTO part of the system but just characterizes the key parameters of the ETO options (among them performance and cost). As a consequence, if the customer accepts the offer, the supplier must design in detail every ETO option, their integration and production process before launching production. This is where the risky point lies as explained by [3]. As the offer has been submitted and accepted with given performance, cost and due date, without a detail study of the ETO options, the supplier takes the risk of not being able to match what he has promised and sold. This means that the final delivered system might be more expensive and/or longer to produce than expected.

We assume that the offer elaboration is achieved thanks to a concurrent system/process configuration [4] activity supported by:

- a system configuration software in order to configure the CTO part of the system that has some kind of a "design gate" for ETO enabling the user to capture the rough ideas about the solution relevant to ETO options [5], [6].
- a delivery process configuration software in order to configure the design activity (for ETO options) and the production activities (for the whole system, from sourcing, assembling up to installing and test) [7].

The risk, previously characterized, is therefore attached to the delivery process. Therefore, following the system/process configuration activity, a second activity is concerned with what we call the risk configuration relevant to the delivery process as shown in Fig.1.

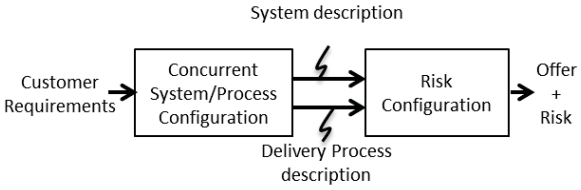


Fig. 1 - Offer elaboration and risk configuration

Similarly to knowledge fundamentals relevant to configuration key ideas [8] we consider that it should be possible: (i) to gather risk knowledge and risk processing knowledge in a knowledge base, as a kind of generic model, and (ii) to propose a knowledge interactive process that allows to support risk configuration for a specific risk. In this purpose, the rest of this article goes as follows: in a second section, we identify the knowledge involved in risk configuration. In the third section, we define the risk generic model and the risk configuration problem. In the fourth section, the first ideas relevant to knowledge retrieval queries that can support risk configuration when elaborating offers are proposed.

[1] Industrial Engineering Center, Toulouse University – IMT Mines Albi, Albi, France
[2] Laboratoire de Génie de Production, Toulouse University – INP-ENIT, Tarbes, France
[3] ESTIA Recherche - Bidart - France
corresponding author e-mail: michel.aldanondo@mines-albi.fr

## 2 RISK IDENTIFICATION IN DELIVERY PROCESS

According to [9], risk processing in new product development projects can go as follows. The delivery process is considered as the main input. In our situation, the delivery process gathers a set of tasks as: finalize design, source sub-systems and/or raw-materials, manufacture, assemble, test and deliver. Each of these tasks is analyzed by a risk expert who identify for each task: (i) the negative event that can be associated to the risk with its occurrence probability [17], (ii) the impacts as modifications of the cost or duration of some tasks, (iii) the possible corrective or preventive actions, in order to counter the risk and (iv) the impacts reductions and/or risk probability reductions as a result of these corrective or preventive actions. Our goal is so far to establish a knowledge model and an interactive process to support the person in charge of these identifications that we call risk configuration.

With regard to risks in the customer-supplier relationship domain, most of the works are based on marketing approaches, logistics issues or supplier selection [10] or [11]. We retain the work of [11] because: (i) they propose a classification of the risks according to the type of customer-supplier relationship, (ii) they clearly differentiate the risks "buyer" and "seller" and (iii) they stress the need to consider the supplier's point of view. Our work is fully in line with this last point. Regarding the risks in offer elaboration, we did not find any work addressing the problem as we formulated it. On the other hand, there is more works and normative elements concerning the risks in project management [12]. We are part of this workflow and especially in the continuity of the approaches proposed in [9] and [13], where a notion of risk processing strategy is proposed.

Regarding the modeling and exploitation of risk knowledge to support offer elaboration in customer-supplier relationship, the work is much rarer. Some works exist in civil engineering [14] or in information system project [15]. As far as we know, only [14] proposes knowledge modeling elements for risks. We join in this type of contribution and will propose elements of knowledge models.

As seen previously, the delivery process logically constitutes a first input of risk configuration, because the risks and their impacts and treatments are defined with respect to the tasks of this delivery process. We describe this process as follows:

- The delivery process is associated with a system that is the subject of an offer. A same system can be associated with several delivery processes, in order to compare process variants.
- A process is broken down into tasks linked by precedence constraints. Each task i is performed by a key resource (resource needed to perform it) and is characterized by a cost, ci, and a duration, di, (other metrics could be defined) as shown in Fig. 2.

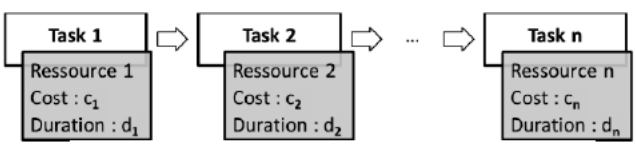


Fig. 2 – Process subject to risk configuration

These data are not sufficient to perform the risk configuration. Other data relevant to the engineered system or the general context of the offer have to be taken into account by the expert. System characteristics impacting the risk configuration can be for example: the complexity or the size of the system, the maturity of the technologies employed, the reliability of the components used, etc. General context of the offer impacting the risk configuration can be for example: the importance of the customer, the recurrence of the demand, the workload of the supplier, etc.

The risk expert has the knowledge about these characteristics and their impacts on the tasks of the delivery process. They strongly modulate the values of risk probabilities, impacts and impact reductions. We propose to describe these characteristics using the triplet:

(1) Conceptual element, for example: crane system, engine component, offer, customer, etc.
(2) Attribute describing the conceptual elements, e.g. crane complexity, engine maturity, offer recurrence, importance of customer, etc.
(3) Value of the attribute describing the concept, for example, very strong / strong / weak / very weak, high / medium / low.

Of course for example : (i) an offer of a high complexity crane with a low maturity engine will be more risky and (ii) risk processing and impact reductions will be stronger if the customer is important.

As a conclusion two inputs will be used by the expert, the delivery process description and the conceptual elements attributes that impacts risk configuration.

## 3 RISK CONFIGURATION AND GENERIC MODEL

We consider a risk as a pair (task, event), meaning that the event that occurs during this task correspond with the risk. This is questionable but makes it possible to dissociate the analysis of the consequences of the same event on different tasks. For example, this allows the event "Snowfall and blocked road" to be analyzed differently, depending on whether it occurs during a "Component sourcing" task or during a "Final delivery to customer" task. Remark also that a same task can be the source of several risks. For example, a "Finalizing the design" task can be subject to two risks "Task more difficult than expected" and "Key resource unavailable".

A risk is associated with a set of impacts. An impact is defined for a single risk with:

- the impacted task, i.e. the task itself, or some others tasks of the delivery process,
- the nature of the impact: cost or duration (or other metrics),
- the method of calculating the impact: either additive (+) or multiplicative (*),
- the value of the impact.

For example, the risk " Finalizing the design task more difficult than expected" can have two impacts, one on the "Finalizing design" task by adding two additional days to its duration and the

other on the "Assembling and testing" task by multiplying its cost by 1.5. Let's note that a given task can be the object of several impacts resulting from different risks Ri.

A risk is characterized by the probability of occurrence [0, 1] of the associated event, on the studied task [17]. This probability can be link to the general context of an offer and some system characteristics. For instance, the probability of occurrence of the risk " Finalizing the design task more difficult than expected" will be higher for a new customer and a very competitive market, rather than for a regular customer with a stable market.

A risk can be mitigated thanks to several preventive and corrective actions. Each corrective or preventive action is a task related to the tasks of the delivery process by precedence constraints. A corrective or preventive action can be associated with different local strategies of the same risk. Each corrective or preventive action is characterized by duration and cost. For example, a preventive action to mitigate the risk "Task more difficult than expected" on the "Finalizing the design" task can be to "train the designers on specific software or design method".

The generic model of a risk is presented in Fig. 3.

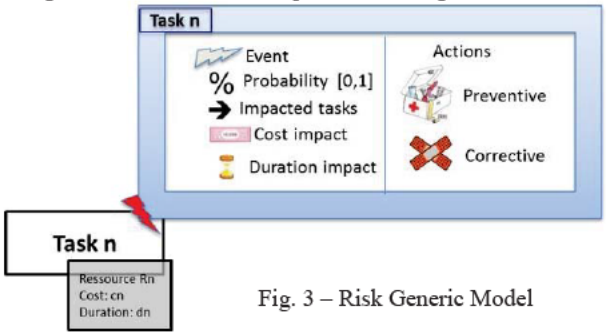


Fig. 3 – Risk Generic Model

In a similar way of product configuration [18], we therefore define **Risk Configuration** as:

- Given a general context of an offer, some system characteristics and one task of the delivery process,
- Given that a generic model of a risk gathers:
  - i. a set of events,
  - ii. a set of possible impacts,
  - iii. a set of impacted tasks,
  - iv. a probability of occurrence,
  - v. a set of preventive actions,
  - vi. a set of corrective actions,
- Given a set of constraints linking general context, system characteristics and risk model items,
- Risk configuration consists in (1) characterizing a risk and its impacts on the delivery process, and (2) finding the set of preventive and corrective actions to be added to the delivery process to mitigate the risk.

This configuration activity can be done either by using constraints satisfaction problem or by the exploitation of past cases stored in a database thanks to case-based reasoning (object of this short paper).

For instance, the instantiation of the risk "Task more difficult than expected" on the "Finalizing the design" is presented in Fig. 4.

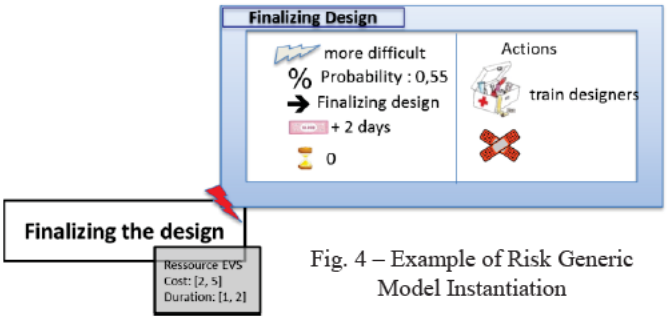


Fig. 4 – Example of Risk Generic Model Instantiation

## 4 AIDING RISK CONFIGURATION WITH CASE-BASED REASONING

Based on the principles of reasoning by analogy or case-based reasoning [16], our idea is to store in a case base all the information describing the effective realization of past delivery processes (these processes correspond to offers accepted by the customer, the process would not be realized). Then, in the presence of a currently studied offer, the objective is to look for similar items in the case base and retrieve the risk information pertaining for risk configuration in order to provide it as a suggestion to the person in charge of the risk configuration. Three examples of queries relevant to risk, risk impact and risk strategies are proposed.

The first query proposes, for a given task, possible risks with probabilities as follows:

- for a given task of a current process offer,
- select in the case base, the risk associated to the tasks similar or identical to the given task,
- select from the selected risk events, the risk event and its probabilities with conceptual element attributes same or similar to those of the current offer, propose these results to the person in charge.

The second query proposes, for a given risk, possible impacted tasks with impacts values as follows:

- for a given pair (task, event) of a current process offer,
- select in the case base, the impacted tasks affected by a same or similar pair (task, event),
- select from the selected impacted tasks, the impacted tasks and their impacts values which are the same or similar to those of the current offer, propose these results to the person in charge.

The third query proposes, for a given risk, possible strategies with corrective/preventive actions as follows:

- for a given pair (task, event) of a current process offer,
- select in the case base, the impacted tasks affected by a same or similar pair (task, event), in a similar way as for the second query,
- select in the case base, the risk strategies with their corrective/preventive actions which are the same or similar to previously founded impacted tasks,

- select from selected risk strategies, corrective/ preventive actions with conceptual element attributes same or similar to those of the current offer, propose these results to the person in charge.

These three queries are given as examples. Other one can be established. They provide a strong support to the person in charge of risk configuration in the sense that they avoid him to rely only on his own knowledge or risk expertise.

## 5 CONCLUSION

The goal of this short article was to provide first elements in order to set-up to a knowledge-based support system for risk configuration when answering tenders or direct customer demands. Risk configuration knowledge has been identified, a generic model of risk was provided as well as a risk configuration definition. Some examples of queries to assist risk configuration by the use of past cases have been also provided.

Two main interests of this proposition are:

- to support and to give confidence to the risk expert suggestions,
- to allow being less human expert dependent.

In a more operational and industrial level, another key interest of this proposition is to allow companies:

- to reduce the level of expertise required to engineer conventional risks (with a junior risk expert for example)
- to leave more time to senior experts to focus on unconventional risks (new risks or critical risks, for example).

Given all these elements and according to the approach advocated in [9] that proposes to use a discrete event simulator, the expert is now able to:

- configure the risks of a given delivery process taking into account the offer context and system characteristics,
- simulate the delivery process with all possible combinations of risk occurrences taking into account corrective and preventive actions
- and evaluate for each combination of risks, the relevant metrics and probability of occurrence.

These simulations allow the expert to evaluate all the scenarios from the worst one (with all the risk occurrences and no relevant preventive and/or corrective actions) to the best one (no risk occurrence and no additive expenses or loses of time due to preventive or corrective actions), and to give for each of them the probability of occurrence and the value for the relevant metrics (cost and duration). Therefore, in this study, this first level of risk configuration/simulation allows the expert to know "what could happen if things don't go as they should with and without preventive and corrective actions".

As far as we know, we did not find any scientific work relevant to this problem. We are at the present time beginning to prototype and test this knowledge base system with four companies from industrial and service sectors. The next issue is to add some rule-based decision aiding, assuming that some generic risk configuration rules can be extracted from the case base.

## ACKNOWLEDGMENTS

The authors would like to thank all ANR OPERA partners and the French ANR agency for work funding.

## REFERENCES

[1] Rivest L., Desrocher A., Brie A., 'Adaptive generic product structure modelling for design reuse in engineer-to-order products', *Computers in Industry*, **61**, 53–65 (2000).

[2] Markworth S., Hvam L., 'Understanding the impact of non-standard customisations in an engineer-to-order context: A case study', *Int. J. of Production Research*, (2018).

[3] Sylla A., Vareilles E., Coudert T., Kirytopoulos K., Aldanondo M., Geneste L, 'Readiness, feasibility and confidence: how to help bidders to better develop and assess their offer's, *Int. J. of Production Research*, **55** (23), 7204–7222, (2017)

[4] Pitiot P., Aldanondo M., Vareilles E., Gaborit P., Djefel M., Carbonnel S., 'Concurrent product configuration and process planning, towards an approach combining interactivity and optimality', *Int. J. of Production Research*, **51** (2), 524-541, (2013).

[5] Sylla A., Guillon D., Vareilles E., Aldanondo M., Coudert T., Geneste L., 'Configuration knowledge modeling: How to extend configuration from assemble/make to order towards engineer to order for the bidding process', *Computers in Industry* , **99**, 29–41, (2018)

[6] Vareilles E., Aldanondo M., Gaborit P., 'Evaluation and design: A knowledge-based approach', *Int. J. of Computer Integrated Manuf.*, **20** (7), 659–653, (2007)

[7] Zhang L., Vareilles E., Aldanondo M., 'Generic Bill of Functions, Materials, and Operations for SAP 2 Configuration', *Int. J. of Production Research*, **51**(2), 465–478 (2013)

[8] Felfernig A., Hotz L., Bagley C., Tiihonen J., *Knowledge-Based Configuration From Research to Business Cases*, Ed Morgan Kaufmann (2014).

[9] Marmier F., Gourc D., Laarz F, 'A risk oriented model to assess strategic decisions in new product development projects', *Decision Support Systems*, **56**, 74–82 (2013).

[10] Thun J.H., Hoenig D., 'An empirical analysis of supply chain risk management in the German automotive industry', *Int. J. of Production Economics*, **131**(1), 242-249 (2011).

[11] Hallikas J, Puumalainen K., Vesterinen T., Virolainen V., 'Risk-based classification of supplier relationships', *J. of Purchasing and Supply Manag*., **11(**2-3), 72-82, (2005).

[12] ISO 31000, *International Standards for Business, Risk Management* – Principles and Guidelines, (2009).

[13] Fang C., Marle F., 'A simulation-based risk network model for decision support in project risk management', *Decision Support Systems*, **52**, 635–644, (2012).

[14] Yildiz A.E., Dikmena I., Birgonul M.T., Ercoskunb K., Alten S., 'A knowledge-based risk mapping tool for cost estimation of international construction projects', *Automation in Construction,* **43**, 144–155, (2014).

[15] Alhawaria S., Karadshehb L., Talet A.N., Mansoura E., 'Knowledge-Based Risk Management framework for Information Technology project', *Int. J. of Information Manag*., **32**, 50 – 65, (2012).

[16] Aamodt, A., Plaza, E., 'Case-based reasoning: foundational issues, methodological variations, and system approaches', *AI Communications*, **7**(1), 39–52, (1994).

[17] Hillson, D. & Hulett, D. T, *Assessing risk probability: alternative approaches*. Paper presented at PMI® Global Congress 2004—EMEA, Prague, Czech Republic. Newtown Square, PA: Project Management Institute (2004).

[18] Mittal S. and Frayman F., T*oward a generic model of Configuration Tasks*, Int. Joint Conferences on Artificial Intelligence, 1395-140, (1989).

# How to deal with Engineering-to-Order Product/System Configuration?

**Abdourahim Sylla**[1, 2] and **Delphine Guillon**[1, 3] and **Rania Ayachi**[1, 2] and **Elise Vareilles**[1] and **Michel Aldanondo**[1] and **Thierry Coudert**[2] **and Laurent Geneste**[2]

**Abstract.** This paper considers the configuration of physical systems in a business to business environment (machine tool, aerospace equipment, cranes …). In this kind of business, knowledge-based configuration software are frequently used when dealing with Assemble/Make-To-Order or (Configure-To-Order (CTO)) situations where the entire customer's requirements can be fulfilled with standard systems. However, in Engineer-To-Order (ETO) situations where non-standard systems must be designed in order to fulfill the entire customers' requirements, existing knowledge-based configuration software cannot be used. In fact, the configuration hypothesis state that all configured systems are assembled from standard sub-systems and components. The aim of this paper is therefore to investigate how the existing products/systems configuration hypothesis, problems' definitions, and models can be modified or adapted in order to allow the use of configuration software in ETO situations. In this purpose, first, the main differences between standard and non-standard systems are analyzed. Then, six cases of systems configuration that differentiate CTO from ETO are identified and discussed. Finally, some Constraint Satisfaction Problems (CSP) based modeling extensions are proposed to allow the use of configuration software in these situations.

## 1 INTRODUCTION

The current economic environment is characterized by the increasing demand for personalized systems from the client companies. In addition, the requirements on the performances, costs and delivery times of the systems are increasingly constrained. Therefore, in order to propose relevant systems solutions to the client companies, the supplier companies have to design customized systems in a very short period while optimizing time and resources involved in the design process [1],[2],[3]. In this article, a system is considered as a set of sub-systems that are integrated following the system architecture.

The design of a system that fulfils the customer' requirements is carried out using three kind of knowledge: (i) the knowledge about the customer's requirements that are the source of the design problem, (ii) the knowledge about the potential systems solutions relevant to these requirements, and (iii) the knowledge about the design methodology [4]. Depending on the availability of these three kinds of knowledge, two types of industrial situations are encountered by the suppliers when designing systems solutions relevant to the customer's requirements [4]: (i) Configure-To-Order (CTO) which gathers both Assemble-To-Order and Make-To-Order industrial situations, and (ii) Engineer-To-Order (ETO).

In Configure-To-Order (CTO) contexts, the relevant knowledge necessary for the design of systems solutions that fulfill the customer's requirements are available. The design of a system in this case, consists in choosing systems solutions that correspond to the requirements [4],[5]. This problem refers to the configuration problem also called customization [7]. In this situation, all possible systems solutions that are relevant to the customer's requirements have been totally designed or predefined. The supplier has just to choose one system solution to propose to the customer. This configuration problem is encountered in many industries, including in the automotive, aeronautics or the micro-informatics sectors [8],[9]. In fact, most of the time, the systems or sub-systems solutions must be selected from a huge number of types or variants to meet specific customer' requirements [8],[9]. Knowledge-based configuration software is very often used by the suppliers to rapidly configure systems that fulfill the customers' requirements.

In Engineer-To-Order (ETO) situations, some modifications or adaptations must be performed on existing systems solutions in order to design systems that fulfil the entire customer's requirements [6]. For example, a customer wants a crane system composed of two sub-systems: a jib of 7 meters long and a tower of 10 meters high. The existing solutions cover the tower sub-system. However, until now, the supplier has only designed jibs of 5 and 9 meters long. Therefore, a jib of 7 meters long must be designed and integrated to the other sub-systems solutions in order to fulfil the entire customer's requirements. Depending on the extent of the design activities necessary to define a system solution that satisfy the entire customer's requirements, some authors and practitioners speak of "light" and "heavy" ETO. In any ETO situations, the existing configuration software cannot be used to configure the entire system. Indeed, the configuration makes the assumption that a system is assembled or defined from sub-systems and components that have been totally designed or predefined. The assembly mode of the sub-systems and components is also predefined [10],[8],[9]. As a consequence, some companies use configuration software to design the predefined parts of the system. The other parts are defined manually or using other tools such as Computer Aided Design (CAD) [11],[12],[13]. This results in additional time, resources and efforts in the design process.

In [18], the authors introduced the concepts of open configuration. They stressed out that one of the characteristics of open configuration is the ability to integrate components and

[1] Centre de Génie Industriel, Université de Toulouse / IMT Mines Albi, France, email: firstname.lastname@mines-albi.fr
[2] Laboratoire Génie de Production, Université de Toulouse / INP ENI Tarbes, France, email: firstname.lastname@enit.fr
[3] ESTIA Recherche, Ecole Supérieure des Technologies Industrielles Avancées, France, email: lastname@estia.fr

constraints that are not completely predefined during the configuration process. They presented some example application domains of open configuration. However, the aspects related on how to extend configuration principles towards ETO industrial situations have not been addressed.

The aim of this article is to investigate how the existing configuration hypothesis, problems' definitions, and models can be modified or adapted in order to extend the use of configuration software towards ETO industrial situations. In the section 2, relevant products/systems configuration background, including problems definitions and Constraint Satisfaction Problems (CSP) knowledge modelling, are recalled. In section 3, the main differences between standard and non-standards systems are analyzed. Then, six cases of systems configuration that differentiate CTO from ETO are identified and discussed. Some Constraint Satisfaction Problems (CSP) based modeling extensions that consider the six cases of systems configuration in ETO situations are also proposed.

# 2 PRODUCT/SYSTEM CONFIGURATION IN CTO SITUATIONS

## 2.1 Configuration problem definition

Since the first configuration problems defined by Mittal [14], many products configuration problems have been defined in the scientific literature [8],[9],[15]. According to the problems, different aspects of a product are considered, especially the physical, descriptive, and functional aspects [8],[9],[15]. Among all these definitions, we consider the key elements proposed in [14],[15]. They are presented as follows:

Hypothesis: a system is considered as set of sub-systems
Given:

- each system or sub-subsystem is characterized with a predefined set of attributes which have predefined domains,
- the attributes can be either descriptive (length, power for instance) or key performance indicators such as the cost,
- the sub-systems that have the same characteristics constitute a family of sub-systems,
- the possible combinations or assembly of sub-systems and/or attributes values are predefined with a set of constraints,
- a customer's requirements corresponds to the selection of a sub-system or attributes values.

Objectives: The configuration consists in finding at least one set of sub-systems that satisfy all the constraints and customer's requirements.

As you can see in this configuration problem definition, only systems and sub-systems that have totally been designed or predefined are considered. This is the common point between the configuration problems and models encountered in the scientific literature. They all assume the following hypothesis [8],[9],[14],[15],[16],[17]: (i) a configured product or system is assembled from predefined sub-systems or components, and (ii) the assembly mode is also predefined. As a consequence, these definitions and models are not suitable to the ETO situations where some sub-systems are not totally designed or predefined.

In this article, in section 3.1, we propose some adaptations of this definition to the ETO situations. In the section 3.2, we introduce the CSP-based modelling framework that is used to model systems configuration knowledge. We also present an example of system configuration in CTO situations.

## 2.2 CSP based knowledge modelling

In the scientific literature, the CSP (Constraint Satisfaction Problem) is the most commonly formalism used to formalize configuration knowledge. It gathers three elements: (i) a set variables, (ii) a finite domain for each variable, and (iii) a set of constraint that establishes relationships between variables [19]. Referring to the configuration problem previously defined, a CSP-based configuration model is defined as follow [7],[13],[15]: each sub-system family and each attribute is associated to a variable. A specific sub-system or attribute value is then a value in its corresponding variable domain. The constraints are used either to specify acceptable combinations of sub-system solutions and/or attribute values. For example, in the Fig. 1, the sub-system jib is associated to the variable "Jib solution". Its descriptive attributes are associated to the variables "Length" and "Stiffness". The length of the jib has two possible values "5 meters" and "9 meters" which represents its domain. The constraints are represented with the full line. They link the attributes' values to their corresponding sub-systems' solutions. Using this model in a configuration software, if the customer' requirements correspond to these solutions; the supplier can configure rapidly at least one solution that cover all the requirements. However, if the customer's requirements exceed these solutions, the supplier cannot exploit this model in a configuration software to configure a crane system solution that covers all the requirements, even if the supplier is able to design, produce or assemble and deliver that solution.

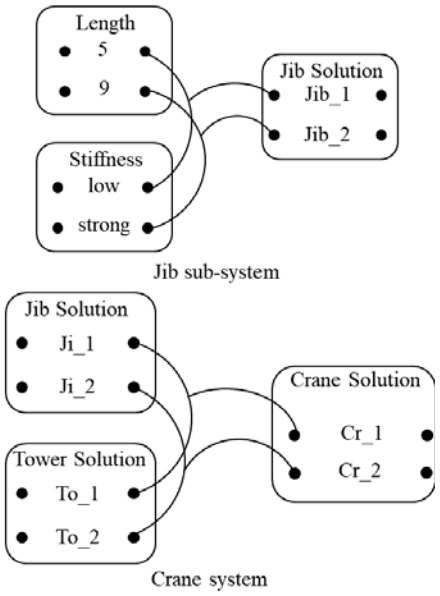


**Figure 1: System configuration model in CTO situations**

In the next section, we propose some modifications or adaptations to the existing configuration problems' definitions and models in order to allow the use of configuration software in ETO situations.

## 3 PROPOSITIONS

In this section, we propose some elements that allow to extend the use of configuration software from CTO towards ETO situations. For this purpose, like Myrodia et al. [12], Aldanondo et al. [15] and other authors, we distinguish : the sub-systems, integrations and systems that have been totally designed or predefined as standard elements, and those that have not been totally designed or predefined as non-standard ones. In the section 3.1, we analysis the main differences between standard and non-standard systems that allow to identify six cases that differentiate the configuration of systems in ETO from CTO situations. In the section 3.2, using a simple example, we show how a configuration model relevant to CTO can be adapted and extended towards ETO situations.

### 3.1 Differences between CTO and ETO

In this section, an analysis of the characteristics of standards and non-standards systems has allowed us to identify the main characteristics which permit to distinguish them. These characteristics rely on: the descriptive attributes of the sub-systems and systems, and the sub-systems that compose the systems. These two elements (descriptive attributes and sub-systems) may: (i) be standard or non-standard, (ii) take standard or non-standard values/instances, and (iii) be the object of standard or non-standard associations/integrations. On this basis, we will talk about standard systems configuration (a configuration in a CTO situation) when all elements, all values or instances, and all associations or integrations are standard. In contrast, for any other case, we will talk about non-standard systems configuration. Thus, the presence of a non-standard feature implies a case of non-standard systems configuration (a configuration in a ETO situation). This analysis has leaded us to identify six cases that differentiate the configuration in CTO from ETO. They represent the different cases of systems configuration in CTO situation. Three cases concern the sub-systems and three relate to the systems. They are presented in Fig. 2 and are described in the following.

The three cases at the sub-system level are:

Case 1: Non-standard association of standard values for the descriptive attributes. This happens when two or more descriptive attributes values that have never been associated together to configure a sub-system have to be associated in order to fulfil customer' requirements. For example, in the Fig. 1, a jib with "5 meters" long and "strong" stiffness is required by a customer.
Case 2: Addition of non-standard values for a descriptive attribute. This happens when a non-standard value must be considered for a descriptive attribute in order to fulfill customer's requirements. For example, a customer wants a jib with "11 meters" long.
Case 3: Addition of non-standard attribute for a sub-system. In this case, a non-standard attribute must be added to configure a sub-system that fulfills customer's requirements. For example, a customer asks for a jib with a specific "shape".

The three cases at the system level are:

Case 4: Non-standard integration of standard instances or solutions for the sub-systems. This happens when two or more sub-systems solutions that have never been integrated together to configure a system, must be integrated to fulfil customer' requirements. For example, the jib "ji_1" and the tower "To_2" must be integrated to fulfil a customer's requirements in the Fig. 1.
Case 5: Addition of a non-standard instance or solution for a sub-system. This happens when a non-standard sub-system solution must be considered for a sub-system in order to fulfill customer's requirements. For example, a customer wants a tower different from "To_1" and "To_2".
Case 6: Addition of non-standard sub-system to a system. In this case, a non-standard sub-system that has never been considered in a system must be added to configure a system that fulfills the customer's requirements. For example, a customer wants a control cabin.

In each of these six cases, all the standard solutions that constitute the diversity of systems (options and variants), formalized in a generic model, do not cover the entire customer's requirements. Non-standard systems must be configured. However, as the knowledge related to these non-standard systems is not formalized in a generic model, they cannot be exploited in a configuration software to configure a non-standard system relevant to the customer's requirements.

Therefore, in order to allow the construction of generic models that gather knowledge related to both standard and non-standard systems, a definition of standard and non-standard system configuration problem is proposed in the following. It includes standard and non-standard element.

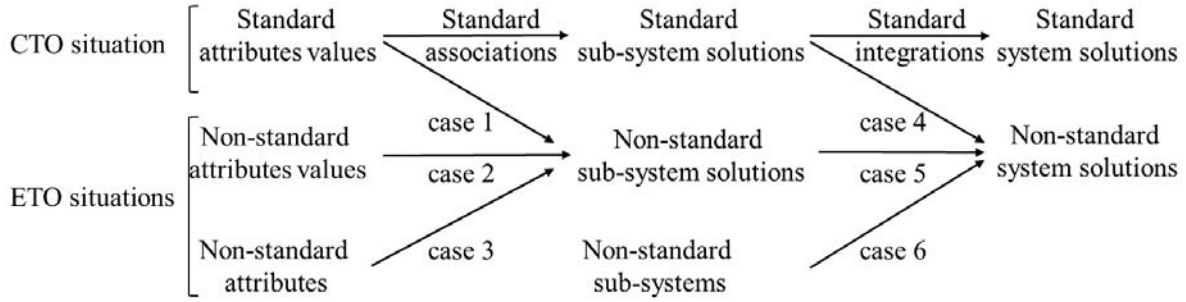


**Figure 2: The six cases of systems configuration in ETO situations**

**Hypothesis:** a system is considered as set of sub-systems;
**Given:**

- each system or sub-subsystem is characterized with a standard or non-standard set of attributes which have standard or non-standards values in their domains,
- the attributes can be either descriptive (length, power for instance) or key performance indicators such as the cost,
- the sub-systems that have the same characteristics constitute a family, they can be standard or non-standard,
- the possible combinations or assemblies of sub-systems and/or attributes values are defined with a set of constraints, the combinations can be standard or non-standard,
- a customer's requirements corresponds to the selection of a sub-system or attributes values.

**Objectives:** The configuration consists in finding at least one set of standard and/or non-standard sub-systems that satisfy all the constraints and customer's requirements.

Based on this definition, in the section 3.2, we propose some modelling approaches that enable to extend existing configuration models relevant to CTO towards ETO situations.

## 3.2 CSP-based Modelling approaches for systems configuration in ETO situations

For each of the six cases of configuration of systems in ETO situations listed in the section 3.1, we have proposed some modifications on the existing configuration models in order to include knowledge related to non-standard elements in the generic models. These modifications include changes to the variables and their domain (the set of possible values), as well as changes to the constraints that bind them. In this article, we only present the extension for the case 1 at the sub-system level and the case 5 at the system level. The same example used for the configuration of system in ETO is used. The model is presented in the Fig. 3. This model is a very simple one. The aim is to show how a configuration model relevant to CTO situation can be modified and extended towards ETO.

At the upper level of the Fig. 3, the sub-system model (case 1) is presented. The same variables as for the configuration model in CTO situation are kept. The main differences are:

- a non-standard sub-system instance or solution "Ji_NS" is added to the domain of the "Jib Solution", this enable the supplier to know that this solution has not been totally designed yet.
- a constraint is added for the non-standard association of standard values; it links the values "5 meters" of the attribute "length", the value "strong" of the attribute "stiffness" and the non-standard solution "Ji_NS";

At the lower level of the Fig. 3, the system model (case 5) is presented. The same variables as for the configuration model in ETO situation are also kept. The main differences are:

- a non-standard sub-system instance or solution "Ji_NS" is added to the domain of the "Jib Solution", it results from the modification made at the sub-system level;
- a non-standard system instance or solution "Cr_NS" is added to the domain of the "Crane Solution"; as for the sub-system, it enables the supplier to know that this system has not been totally designed yet;
- two constraints are added for the non-standard integrations of : "Ji_NS" and "To_1", and "Ji_NS" and "To_2".

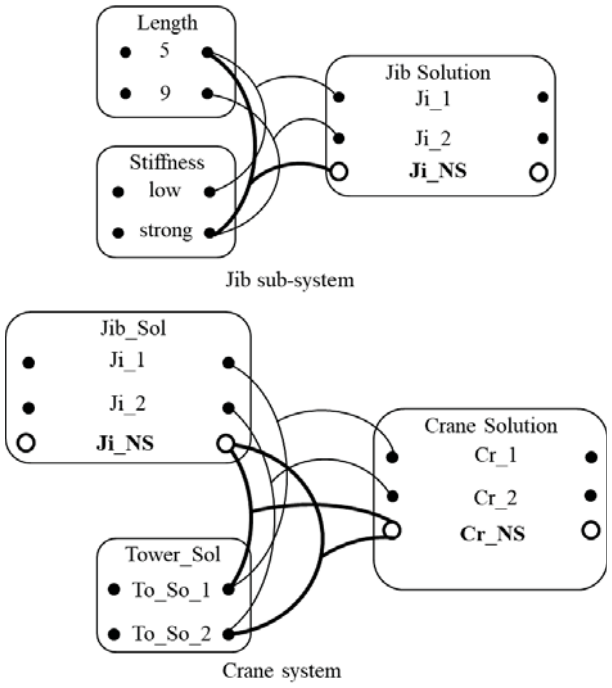


**Figure 3: System configuration model in ETO situations**

For both AMTO and ETO industrial situations, the knowledge necessary to setup the configuration models must be defined by expert teams composed of people from the sales, manufacture, and design departments. The experts must decide on the standard and non-standard systems that can be designed, produced and delivered to a potential customer. This means, with respect to six cases identified in section 3.1, deciding which of the following non-standard aspects can be accepted: (i) combination of standard attribute values, (ii) attribute values (iii) attributes, (iv) integration of standard sub-system solutions, (v) integration of standard and non-standard sub-system solutions and (vi) sub-system. After identifying and validating the knowledge necessary to setup the configuration models for both AMTO and ETO situations, the proposed modelling approach can be used to build configuration models relevant to configure systems in both AMTO and ETO industrial situations.

# 4 CONCLUSION AND FUTURE WORK

In this article, we have studied the configuration of physical systems in the context of business to business environment where a supplier has to propose a system solution to a client company in a very short period while optimizing time, resources and efforts involved. The aim of the article was to propose some solutions in order to extend the use of configuration software from CTO towards ETO situations.

For this purpose, first, we have shown why the existing configuration hypothesis, problems' definitions and models are not adapted for systems configuration in CTO situations. Then, we have analyzed the main differences between standard and non-standard systems. This has allowed us to identify six cases of systems configuration that differentiate the configuration of systems in CTO from ETO situations. The six cases represent the different situations of systems configuration in ETO. This is the main contribution of this article. As far as we know, no scientific work has proposed a formalization of the differences between systems configuration in CTO and ETO situations. Finally, based on these six cases and the configuration background, we have proposed a definition and some CSP (Constraint Satisfaction Problems) modelling approaches for systems configuration problems in CTO and ETO situations. A simple example is used to illustrate the propositions.

As a future research, we intend to test the applicability of our proposals on a larger case of systems configuration. We also intend to extend the configuration of processes relevant to CTO towards ETO.

## ACKNOWLEDGMENTS

The authors would like to thank all ANR OPERA partners for their implications in our research work.

## REFERENCES

[1] M. Krömker, K. D. Thoben, and A. Wickner, 'An infrastructure to support concurrent engineering in bid preparation', *Comput. Ind*., vol. 33, no. 97, pp. 201–208, 1997.

[2] A. Sylla, E. Vareilles, M. Aldanondo, T. Coudert, and K. Kirytopoulos, 'Customer / Supplier Relationship : reducing Uncertainties in Commercial Offers thanks to Readiness , Risk and Confidence Considerations', *in Advances on Mechanics, Design Engineering and Manufacturing*, 2017, pp. 1115–1122.

[3] J. p Cannon and C. Homburg, 'Buyer-Supplier Relationships and Customer Firm Costs', J. Mark., vol. 65, no. 1, pp. 29–43, 2018.

[4] B. Chandrasekaran, 'Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design', *IEEE Expert*, vol. 1(3), pp. 23–30, 1986.

[5] P. Pitiot, M. Aldanondo, E. Vareilles, P. Gaborit, M. Djefel, and S. Carbonnel, 'Concurrent product configuration and process planning, towards an approach combining interactivity and optimality', *Int. J. Prod. Res*., vol. 7543, no. December 2014, pp. 1–18, 2012.

[6] A. Sylla, E. Vareilles, T. Coudert, M. Aldanondo, and L. Geneste, 'Readiness , feasibility and confidence : how to help bidders to better develop and assess their offers', *Int. J. Prod. Res*., 2017.

[7] D. Sabin and R. Weigel, 'Product configuration frameworks- a survey', IEEE Intell. Syst., vol. 13, no. 4, pp. 42–49, 1998.

[8] A. Felfernig, L. Hotz, C. Baglay, and J. Tiihonen, Knowledge-based configuration From Research to Business Cases. 2014.

[9] L. L. Zhang, 'Product configuration : a review of the state-of-the-art and future research', *Int. J. Prod. Res*., vol. 52, no. 21, pp. 6381–6398, 2014.

[10] S. Mittal and F. Frayman, 'Towards a generic model of configuration tasks', *Proc. Elev. Int. Jt. Conf. Artif. Intell*., vol. 2, pp. 1395–1401, 1989.

[11] A. Haug, L. Hvam, and N. H. Mortensen, 'Reducing variety in product solution spaces of engineer-to-order companies : The case of Novenco A / S', *Int. J. Prod. Dev*., vol. 18, no. 6, pp. 531–547, 2013.

[12] A. Myrodia, K. Kristjansdottir, and L. Hvam, 'Impact of product configuration systems on product profitability and costing accuracy', *Comput. Ind*., vol. 88, pp. 12–18, 2017.

[13] A. Sylla, E. Vareilles, T. Coudert, M. Aldanondo, L. Geneste, and Y. Beauregard, 'ETO Bid Solutions Definition and Selection Using Configuration Models and a Multi-Criteria Approach', *in IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, 2017, pp. 1833–1837.

[14] S. Mittal and F. Frayman, "Towards a generic model of configuration tasks," *in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, vol. 2, pp. 1395–1401.

[15] M. Aldanondo and E. Vareilles, "Configuration for mass customization: How to extend product configuration towards requirements and process configuration," *J. Intell. Manuf.*, vol. 19, no. 5, pp. 521–535, 2008.

[16] T. Soininen, J. Tiihonen, M. Tomi, R. Sulonen, and T. Männistö, 'Towards a general ontology of configuration', *Aiedam*, vol. 12, no. 4, pp. 357–372, 1998.

[17] A. Günter and C. Kühn, 'Knowledge-Based Configuration – Survey and Future Directions', *in German Conference on Knowledge-Based Systems*, 1999, pp. 47–66.

[18] A. Felfernig, M. Stettinger, G. Ninaus, M. Jeran, S. Reiterer, A. Falkner, G. Leitner and J. Tiihonen, 'Towards Open Configuration', *in Configuration Workshop*, pp. 89 – 94, 2014.

[19] U. Montanari, 'Network of Constraints: Fundamental Properties and Applications to Picture Processingr', *Inf. Sci*., vol. 7, pp. 97–132, 1974.

# Towards Knowledge Infrastructure for Highly Variant Voltage Transmission Systems

**Mathias Uta** and **Alexander Felfernig**

**Abstract.** The high voltage transmission business uses very mature technical solutions to transport electrical energy over large distances. New developments in the information technology sector are now promising opportunities to revolutionize the traditional processes within the business. In this paper the opportunities to implement a sophisticated knowledge infrastructure to improve the efficiency and quality of high voltage product manufacturers will be outlined. Therefore, possible solutions have been assessed to establish a non-redundant data structure and create an advanced database system architecture with respect to business specific requirements, considering in particular product configurators. Based on the proposed master data system, the possibility to create as well as integrate a knowledge system has been evaluated. Accordingly, the introduction of a global knowledge manager is proposed to organize inquiries of product configurators to expert systems and introduce a company-wide framework for rules and constraints. To assure communication between all parts of the software architecture, the implementation of a universally understandable format is discussed. Finally, possibilities to integrate recommendation system mechanisms into the suggested system architecture are highlighted.

## 1 INTRODUCTION

The market of high voltage transmission business is highly competitive. The differentiation through up to date processes, high quality and fast response times is highly desirable to manage the increasingly complex and fast changing customer requirements. This effect is intensified by general technical developments in measurements initiated by the energy transition from fossil to renewable energy sources.

Due to an increasing share of renewable energy into the grid and their decentralized physical arrangement compared to huge power plants, an extension of the electrical grid has to be provided. Additionally, the weather dependency of the new energy sources intensifies this effect. As a result, the utilities are confronted with a fast changing energy market and need to react to the new energy mixture [1]. Not only a transformation has to be managed on the supplier side but also consumers are starting to adopt new technologies with huge electrical energy demand like electrical cars. Therefore, the only solution to assure a secure energy supply in the future seems to be an extension of the transmission grid [2].

The structure of the electrical grid and the main components are shown in Figure 1. The renewable and non-renewable based power plants convert several energy sources to electrical energy and are connected via transformers to the grid. The overhead lines and the cables transport the energy over large distances to consumers. The switchgears integrate many of these lines on one conductor (busbar) and build nodes in the grid. In case of failures circuit breakers separate the faulty part of the grid to prevent blackouts. Additionally, current transformers and voltage transformers give transmission utilities the possibility to measure the transported energy. The collection of all main parts concentrated in one physical spot is called electric power transmission substation.

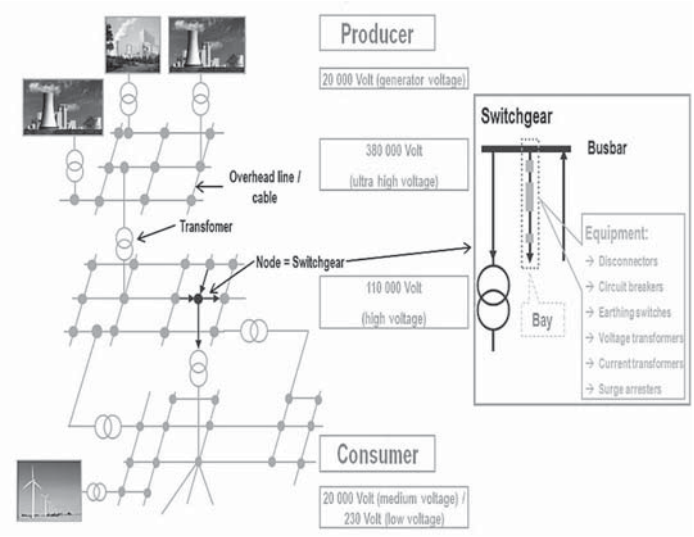


**Figure 1.** Electrical components in the energy grid [3]

Since each of the main modules has to correspond with many diversified customer requirements, the big suppliers in the business such as Siemens, ABB or Alstom have separated their energy transmission business in many smaller segments [4] [5]. Each of these segments is concentrated on one of the main components in the grid. Customers expect suppliers to offer solutions for specifications including a complete electric power transmission substation. As a result, very complex projects with many internal stakeholders have to be managed. This includes the decentralized data storage of in many cases identical information. In each of the involved business segments a unique tool infrastructure is installed and requires varying processes. In particular it is possible that, each business segment uses one or more individual configuration solutions to determine the bill of material derived from individual configuration rules and constraints. These configuration rules are cross segmental on a higher level and do not differ until a finely granulated level of configuration is reached. Furthermore, individual pricing and quotation tools are used which are sometimes integrated with the configuration tool, resulting in so called configuration prizing and quotation solutions (CPQ). As a consequence, very complex processes and redundant actions are

decreasing competitiveness in a cost driven market environment. Additionally, data and knowledge maintenance cause disproportional effort leading to human and hardware resource binding.

To tackle the above described challenges, the introduction of a well-designed data and software infrastructure is necessary. A completely integrated data and software landscape which provides necessary information via interfaces to all quotation related tools should provide synchronization of all involved stakeholders and improve the maintainability of data and knowledge.

In an environment which utilizes several ERP (enterprise resource planning), PDM (product data management) and CRM (customer relationship management) tools, in future one company-wide integrated database should be implemented to deliver one data source for all configuration, pricing and quotation tools. The goal is to enter the information only once into the system so that the characteristic is passed on to all connected business segments and their CPQ tools. In consequence, all related objects rely on the same data input. This assures cross segmental consistency of data in all offer documents, technical descriptions and technical calculations.

Moreover, the integrated representation of data delivers the opportunity to implement consistent connections between objects in form of rules and constraints. This leads to the extension of data to knowledge and should support users by preventing false configurations and incorrect data input, i.e., to contract relevant documents. These relations between the objects need to be non-redundant as per the data itself. Supplemental to the rules and constraints describing simple relationships, interfaces to expert systems capable of very specific and complex calculations need to be established. Mechanical calculations due to earthquake requirements or ferroresonance calculations of the grid are two examples of these expert systems.

Finally, based on the achieved integrated data and knowledge, an analysis of the utilized materials, parameter characteristics and all other objects during the configuration, prizing, and quotation process can act as a basis for a self-learning system to improve the CPQ-tools. This system should be able to recommend technical solutions and parameter input based on previously chosen solutions in distinctive situations as customer specific requirements or special environmental conditions. The user in this scenario still has the opportunity to neglect the recommended solution which is consequently used to improve the recommendation logic.

Overall, a complex system composition consisting of domain specific and integrated databases with direct interfaces to configuration, pricing, and quotation tools has to be established. Further, the user of the quotation tools is supported by knowledge bases inheriting logical connections between relevant parts and an adaptive artificial intelligence which analyses user decisions.

The remainder of this paper is organized as follows. In section 2 we discuss the possibility of a holistic database and point out that different use cases lead to contradicting database types. Conducted from this perception we propose system architecture assuring data consistency for the high voltage transmission business. Based on this data consistency section 3 emphasizes the advantages of knowledge integrity. This is followed by a general survey on an approach to implement a global knowledge manager on basis of the proposed system architecture in section 4. Section 5 suggests a concept of a recommendation system in which the knowledge base is constantly extended using the results of configuration applications used by experts.

## 2 DATABASE EVALUATION

Data modelling plays an important role. To create a ubiquitously applicable database, one has to consider a data structure which allows multiple views and possibilities to manipulate the stored data via interfaces from manifold applications. Graeme C. et al. characterize this situation as follows ‘The data model is a relatively small part of the total systems specification but has a high impact on quality and useful life of the system’ [6]. Data modelling is concentrated on achieving completeness, non-redundancy, and reusability in databases. Additionally, stability, flexibility, and performance have to be taken into account. The implemented database should be capable of modifications and extensions to integrate requirements arising in the future. Nonetheless, the data modeler has to start with the given information to create a so called ‘conceptual schema’ which can be accomplished by different course of action like the process-driven, data-driven, or object-oriented approach to name only the most important ones. Based on the ‘conceptual schema’ a ‘logical schema’ can be deducted by using entity-relationship modelling (ERM) or unified-modelling language (UML) approaches. Finally, a physical design of the database can be created.

Typically used database types are relational databases (RDBs), object-oriented databases (OODBs) and a hybrid form which is called object-relational database (ORDB). Consistency in RDBs is achieved by using normalization methods for tables as proposed in [7]. The objective of OODBs is to create an abstract view on the reality which is easily adaptable to object oriented programming languages. Dependencies between objects are provided by pointers which allow m:n relationship representations. Therefore, OODBs are often adapted for computer aided design (CAD) or technical calculation programs (expert systems) since these programs deal usually with high complexities and many variants [8] [9]. Unfortunately, OODBs lack query performance compared to RDBs, which is why they are still not able to replace traditional relational databases for query focused applications [10]. ORDBs try to combine both approaches to achieve an improved performance and are often adapted by CPQ-tools. Michael Stonebreaker evaluated all three options in [11] and classified them as depicted in Table 1. The table points out ORDBs combine the advantages of RDBs and OODBs in comparison of the most important properties for databases, ‘fast queries’ and ‘complex data management’. But ORDBs implicate some issues as well which is most importantly a low performance in web applications [10].

**Table 1.** A classification of database management systems (DBMS) [11]

| | | |
|---|---|---|
| Query | Relational DBMS | Object-Relational DBMS |
| No Query | File System | Object-Oriented DBMS |
| | Simple Data | Complex Data |

A very detailed analysis of the given requirements is precondition for implementation of the most appropriate database and data structure. Since full integration of the whole value chain, including

PLM (product lifecycle management), ERP (enterprise resource planning), CRM (customer relationship management), configuration, pricing, quotation and technical calculations with expert systems, has to be realized, a solution with one centralized database seems with respect to deviating requirements difficult to implement. Table 2 shows adapted from Alejandro Vaisman's et al. a holistic comparison between these tools and their resulting utilized database types [12].

**Table 2.** Requirement comparison between ERP, CRM, PDM, CPQ and expert systems

| Aspect | ERP / CRM / PDM | CPQ | Expert system |
|---|---|---|---|
| User type | Operators, office employees | Customer, sales employees | Engineers |
| Content | Current, detailed data | Current, detailed knowledge | Current, detailed knowledge |
| Data organization | According to operational needs | According to operational and analysis needs | According to operational needs |
| Data structures | Optimized for small transactions | Optimized for complex queries | Optimized for complex queries |
| Access frequency | High | High | Medium |
| Access type | Read, insert, update, delete | Read, insert, update, delete | Read, insert, update, delete |
| Number of records per access | Few | Many | Many |
| Response time | Short | Short | Can be long |
| Concurrency level | High | High | From medium to low |
| Update frequency | High | From medium to low | From medium to low |
| Data redundancy | Low (normalized tables) | Medium (normalized tables and objects) | Can be high (no normalization methods and inheritance) |
| **Resulting database type** | **Relational database (RDB)** | **Object relational database (ORDB)** | **Object oriented database (OODB)** |

Whereas ERP, CRM, and PDM tools are designed for fast access to distinctive data to satisfy operational queries, CPQ and expert systems are focused on very complex queries that require combinations between many data sources. Additionally, CPQ-tools are facing a high access frequency with expected short response times which leads to an even higher level of complexity in the data structure.

In conclusion, based on these manifold requirements it seems more likely to implement a sophisticated system architecture including several customized databases for each application instead of establishing one ubiquitous database. The system architecture of such a conglomeration of databases is closer examined in the next chapter.

# 3 SYSTEM ARCHITECTURE

The data used along the value chain can be described most likely as 'master data' of a company. Master data is defined as 'data held by an organization that describes the entities that are both independent and fundamental for that organization, and that it needs in reference in order to perform its transactions' [13]. By other means, master data is all the data stored about customers, employees, products, materials, suppliers. The management of this data is called master data management (MDM) and can be provided by mainly four different architectures as shown in Figure 2.

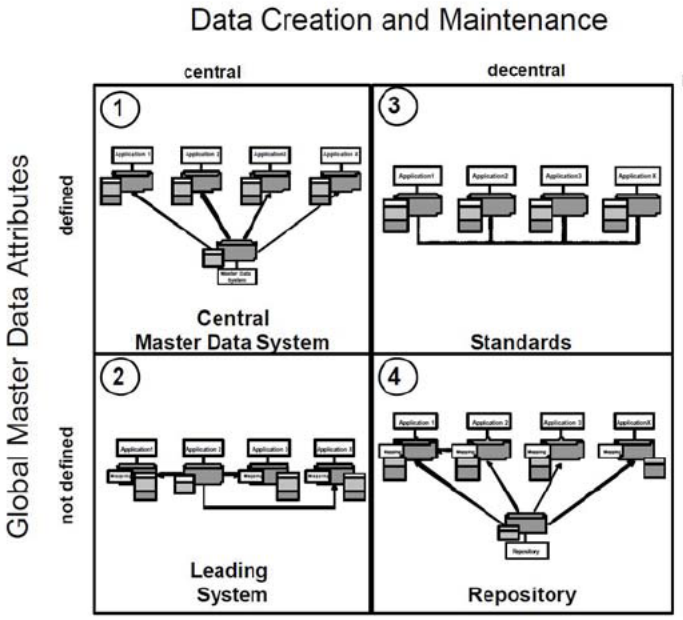


**Figure 2.** Classification of architecture approaches [14]

The architecture in this approach is classified by two major categories:

- Are global master data attributes defined or has each involved application its own master data definitions?
- Where is data created and maintained – in a centralized system or decentralized in each connected application?

In the centralized master data system (1) all global master data attributes are created and maintained in one database. Associated applications receive these attributes directly from this centralized system via identical primary keys whereas additional specific attributes can be defined individually in the single applications. The leading system methodology (2) instead has no global master attributes but a system which is responsible for data creation and maintenance. These attributes are transmitted to connected applications where the primary key of the attribute is mapped to the corresponding attribute in the application.

Decentralized data creation and maintenance approaches on the other hand rely on company-wide standard definitions (3) or an implementation of a repository (4). Company-wide standards do not assure consistency in each dataset of the applications. Instead they focus on the ability to create the same understanding of each attribute which leads to easier manageable comparisons between those datasets. A physical connection between applications is in

this approach not mandatory. The installation of a repository follows the idea of metadata storage to connect applications with each other whenever an interaction is necessary. Therefore, primary keys of all involved applications for a certain attribute are stored in the repository and matched for cross-program query events with each other.

In case of the high voltage transmission business, an implementation of one integrated MDM solution seems, equivalent to the result of the second chapter for databases, not feasible. Two reasons are mainly responsible for implementation of a more complex structure. First of all the structure of the business includes, as already pointed out, several different independent departments responsible for distinct parts of the substation. These departments are not necessarily located at the same location but are distributed over the world to assure an increased satisfaction of locational requirements. Consequentially, one centralized system located, for instance, in Europe will lead to performance issues when it comes to queries from an application located in Asia. Live connections over long distances and many servers are due to the congestion control of the transmission control protocol not advisable [15]. A system which synchronizes independent and asynchronous to operational tasks is therefore preferable. Secondly, each department has sometimes not only one factory but several production sides each with their own portfolio. The main structure of the portfolio is same as per the department. The differences can be found on a more detailed layer of the products as, for instance, in the differentiation of high-end products and cost-optimized products which serve generally the same technical requirements but are differentiated in more sophisticated configuration options as better operation monitoring or increased maintainability of the electrical component. From these considerations follows, a solution incorporating combinations of the previously presented options seem more feasible to encounter the high complexity in the high voltage transmission business and causes the introduction of more than one MDM layer.

Figure 3 shows an option where a centralized master data system is implemented storing master data of one manufacturing location whereas the centralized database is connected to other locations via a repository to assure a synchronization of all business relevant data independent from daily operational tasks. Furthermore, standards across the whole company are defined to realize a homogenous creation of new data since each master data system is allowed to create and maintain its data by itself.

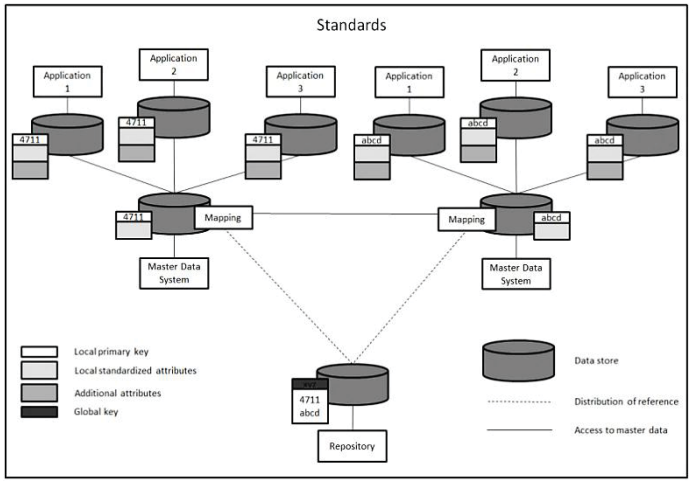


**Figure 3.** Three layer repository - master data system – standards approach

In another scenario a company-wide leading system located in at a distinct place is established to create and maintain all master data. This leading system is additionally used as basis for connected central master data systems installed in distributed factories of the company. These centralized master data systems receive the company-wide master data from the leading system and can be extended with manufacturing side specific master data as basis for connected applications.

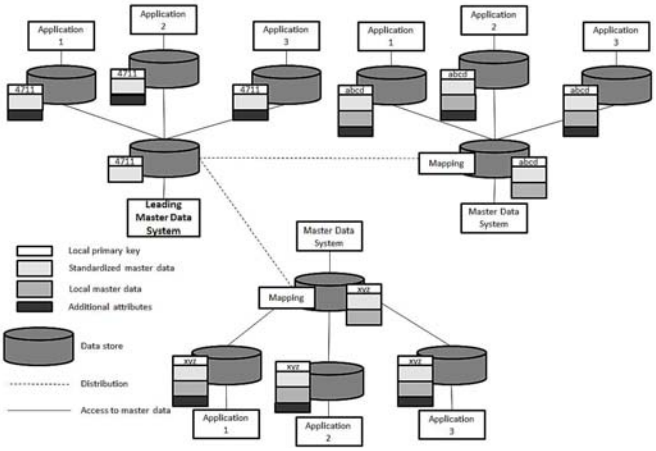


**Figure 4.** Two layer leading master data system approach

Further combinations are imaginable as well and have to be evaluated based on several reference values in further publications. Based on these results and taking all requirements for the high voltage transmission business into account finally a decision on the system architecture can be reached. Another important decision is the selection of the tool used as master data management system. The use of the ERP, CRM or PDM tool is imaginable since the most data is created in these databases. Additionally, the adaption of the CPQ-tool as main input tool for data might also be a reasonable choice.

Up to that point we focused on data structure and system architecture neglecting the value which can be created out of integrated, non-redundant data. The following chapters will discuss the possibilities given by such a dataset.

## 4 KNOWLEDGE INTEGRITY

A well-known use case for master data management systems is the implementation of data warehouses. Operational data is put through an ‘extraction, transformation, integration, and cleansing process (to) store the data in a common repository called data warehouse’ [13]. Data in the data warehouse is in reference to W.H. Inmon saved ‘subject-oriented, integrated, time-variant, nonvolatile (to) support management’s decision making process’ [16]. This means an OODB is implemented that saves data over a long time to enable analysis of developments in the observed business. Besides the data itself metadata – data about the stored data – is stored in the data warehouse. In big organizations, data is not directly used by queries to the data warehouse. Instead, data marts are introduced as an additional layer between the user and the data warehouse. Similar to the already analyzed central master data systems data is transmitted from the data warehouse (central master data system) to the data mart (application database) which fits to the requirements of a specific department. The user of the

data mart is finally able to create reports, perform data analysis and to mine data.

The restriction of this concept is already given in the definition of the data warehouse. The data warehouse should 'support management's decision making process' [16]. The area which besides the management decisions is widely neglected in centralized data and knowledge storage is the configuration and engineering sector. While huge effort is put into preparation of management reports and supporting management decisions, configuration tasks, technical calculations and dimensioning decisions are left completely to engineers and expert departments. This leads to a complete dependency of the company on their team of experts. The problem worsens if the dependence is only on a single expert. Attempts to consolidate the knowledge of engineering experts to more than one employee usually collapse because of budget and time restrictions. The experience of the expert cannot be transferred to other colleagues within in short period of time [17]. It sometimes requires years to reach the level of an expert and could result in loss of knowledge in case the expert retires.

Another issue which is more and more visible to companies arises with the implementation of more than one configuration software. Since the beginning of the new millennium a rising number of companies implement product configurators to treat a phenomenon called 'mass customization' [18]. 'Mass customization' is an oxymoron referring on the one hand to the growing production charges of companies to serve the market and on the other hand the increasing amount of individualization requirements by customers. The configuration technology relies on a subject-oriented, integrated, time-variant, nonvolatile dataset similar to the data warehouse. Combinations of entities and their attributes are configured according to combination and configuration rules to create a producible bill of material.

This technology was immediately adopted by the high voltage transmission business since the modular design of its products is predestinated to realize product configurators. Unfortunately, the degree of standardization in the business does not allow a full switch from CAD (computer aided design) applications to product configuration solutions. Special-purpose solutions define the high voltage transmission market up to fifty percent. The result is an increasing redundancy in the configuration data and knowledge. While the CAD system inherits engineering data accomplished by rules and constraints for the product with a high degree of freedom to support the engineering expert, the product configuration application is designed for sales people or even for the customer and includes a separated engineering dataset complemented by even more sophisticated rules, constraints and methods to prevent a wrong configuration. Nevertheless, a redundancy of engineering data and knowledge is introduced; leading to a high maintenance effort and complicated processes whenever the research and development department (R&D) releases technical innovations or the product lifecycle management department (PLM) disables the use of certain parts of the product.

Based on these evaluations, a centralized engineering database has to be implemented as foundation for all product configurators. PDM systems fortunately inherit all relevant product information and can, as the major tool for PLM and R&D departments, be utilized as a centralized engineering database. The result is a database valid for all configuration applications which are no longer maintained decentral by each product configurator but are centrally maintained by the product responsible departments. The technical realization of this requirement can be established by introducing interfaces between the product configurators and the PDM system. However, with this step an integrated data administration can be introduced but the knowledge administration is still redundantly managed in each configuration tool.

To achieve knowledge integrity - a major target formulated in the visions at the beginning of this paper - we propose steps connatural to the actions necessary to reach data integrity [7] [8] and which are also related to frameworks as formulated in [19]. First a 'conceptual schema' of the knowledge base has to be created using the object-oriented approach since several connections between the given objects have to be considered. Data in the PDM system is most likely stored, as analyzed in chapter 2, in RDBs and has to be addressed to objects in an OODB or ORDB. These objects have to be connected via rules and constraints on a level applicable for all product configurators to implement a 'logical scheme'. Therefore, normalization methods comparable to data normalization need to be formulated. In other words, a framework for product configurators needs to be established 'making the common parts common' [19] and making general rules and constraints applicable for all configuration tools to prevent redundant code implementation. Due to the generalization of rules and constraints on a higher level, a non-redundancy of the knowledge in the separated specialized configuration solutions is achieved and therefore, a centralized maintainability of the knowledge. Redundancy detection algorithms as proposed in [20] can consequentially be applied under these conditions to assure continuous knowledge integrity. Furthermore, the knowledge of experts can be transferred in incremental steps into the knowledge base avoiding redundancy in several tools and decreasing dependency on single persons in the company. Finally, a 'physical scheme' of the system architecture has to be created and is investigated in more detail in the next chapter.

## 5 A GLOBAL KNOWLEDGE MANAGER

An initial approach to set up a system architecture analog to the requirement formulated above was made in the paper 'On Knowledge-Base System Architectures' and is illustrated in Figure 5 [21]. The paper introduced a unit called global knowledge manager (GKM) to centrally handle inquiries by other knowledge-based systems and data type processors (e.g. traditional databases). To assure common semantics a translator or interface was proposed. A request incoming to the GKM is first scheduled and optimized by using the GKM's knowledge base and information delivered by the source of the request. The result is an access plan stored in the GKM's internal database. The monitor/interpreter uses this plan to process the request by applying the GKM's knowledge base rules, constraints and methods. The outcome is returned to the original source using common semantics.

This approach presumes a fully centralized inference of all inquiries. Any information in the system is gathered in the GKM, scheduled and interpreted. In consequence, a centralized expert system is introduced responsible to create and maintain all business relevant rules, constraints and methods irrespective of the complexity and domain of the request. A configuration task to calculate the mechanical forces on a circuit breaker in case of an earthquake would as well be processed as a simple request for earnings before interest and taxes (EBIT) calculation. The consequence would be a very complex set of knowledge

representations neglecting all domain specific requirements. Therefore, a master data and master knowledge approach assuring consistency between all connected tools might be a more feasible solution.

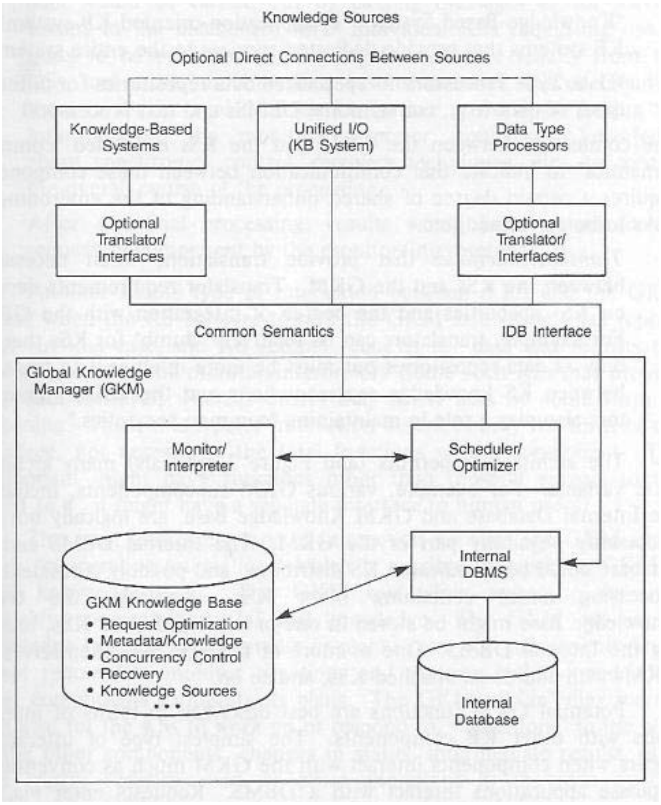


**Figure 5.** Architecture with global knowledge manager [21]

Figure 6 shows, based on the system architecture presented in Figure 3, an overview of the system architecture we propose to provide the requirements of an easy maintainable, integrated system for the high voltage transmission business. A company-wide standard or framework for creation of data, rules and constraints is defined to assure semantic consistency between all locations. The master data system is accomplished by the global knowledge manager to facilitate data and knowledge integrity in one location. While the master data management part is responsible for the data integrity of all databases, the global knowledge manager is comparably responsible for knowledge integrity in all connected configuration applications and expert systems. Additionally the global knowledge manager organizes special calculation inquiries by CPQ-tools to expert tools. Therefore, a universally understandable format (e.g. extensible markup language – XML) is introduced to enable communications between the GKM and all connected tools. Knowledge integrity during operation is assured by redundancy detection algorithms considering all knowledge bases of the software infrastructure. Additionally, a repository is tracking comparable data, rules and constraints of the separately operating locations by defining company-wide primary keys. Changes in the master data and global knowledge management system are synchronized asynchronous to operational inquiries to maintain consistency between all local data and knowledge sets by utilizing the repository primary keys. The behavior of the system is illustrated in more details by the two following examples.

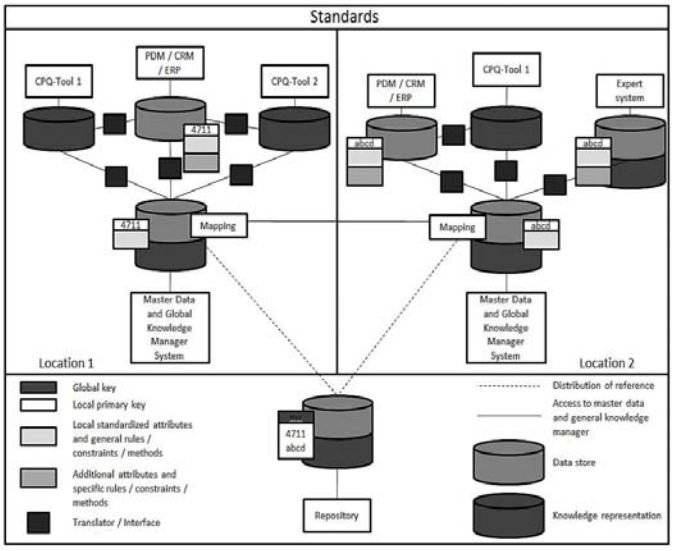


**Figure 6.** Three layer repository – global knowledge management – standards approach

In Location 1, which is in shown in Figure 7, a system architecture including two CPQ-tools and the usual databases is established. CPQ-tool 1 is responsible for very standardized configuration tasks while CPQ-tool 2 is an expert tool to configure specialized customer requirements. Simple if-then relationships which generally describe a part of the substation are maintained in the GKM. For instance: 'If product A is chosen then the cross section of the rectangular conductor cannot exceed 2400 mm².' The information about the product is received via an interface to the PDM database and is maintained in a fast accessible RDB database. In the GKM two objects (product A and rectangular conductor) are described by parameters (cross section, ambient temperature, body temperature, rated current) matched to the information delivered by the PDM database and are connected via the above mentioned relationship. This general rule is accomplished by a more specific rule in CPQ-tool 1 by the following relationship: 'If the rated current exceeds 2500 A at a ambient temperature of 35 °C then a rectangular aluminum conductor needs at least a cross section of 2400 mm² to not exceed 65°C body temperature.' [22].

A user of CPQ-tool 1 is in consequence not allowed to choose product A if the rated current exceeds 2500 A and an ambient temperature of 35°C is given. Contradicting to this rule, an expert in CPQ-tool 2 is not restricted by this constraint. An expert could decide to use product A if the current exceeds 2500 A at a ambient temperature of 35 °C knowing that the conductor is allowed to exceed 65 °C if the conductor is not touchable by humans. This expert knowledge includes besides product knowledge as well knowledge about spatial constraints and human interactions with the product and is with its complexity not easily describable in a non-expert configuration tool. Nevertheless both configuration systems will lead to a correct solution. CPQ-tool 1 will lead to the costlier product B but is operable by the customer herself while CPQ-tool 2 with a higher degree of freedom will deliver a cheaper solution of the configuration task but needs expert knowledge. Therefore, two options for the customer are established, a slightly costlier but very fast configuration or a very accurate solution with a more time consuming configuration.

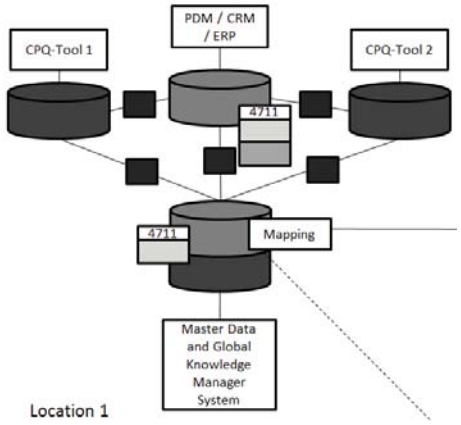


**Figure 7.** Multiple CPQ-tool system architecture

Location 2 is in detailed visible in Figure 8 and includes the standard databases, CPQ-tool 1 and an expert system. The previously mentioned configuration task to calculate the mechanical forces on a circuit breaker in case of an earthquake is given. This calculation task requires information which is not stored in the PDM database or any other standard available master data. Additionally, very specific calculation methods have to be applied not handled in the CPQ-tool. An expert system is necessary to solve the calculation task. The calculation inquiry is sent from the CPQ-tool via the universally understandable format to the GKM that interprets schedules and finally processes the request to the correct expert tool. In consequence, the GKM inherits besides generalized rules and constraints also information about the expert tools in the system and administrates access to them. The inquiry is handled by the expert system using the information received via the interface and by utilizing additional information like specific knowledge on the occurring earthquake forces in the concerned region to calculate the forces on the circuit breaker. The result is processed back to the GKM and further to the CPQ-tool where the resulting values trigger rules and constraints to decide which circuit breaker has to be chosen.

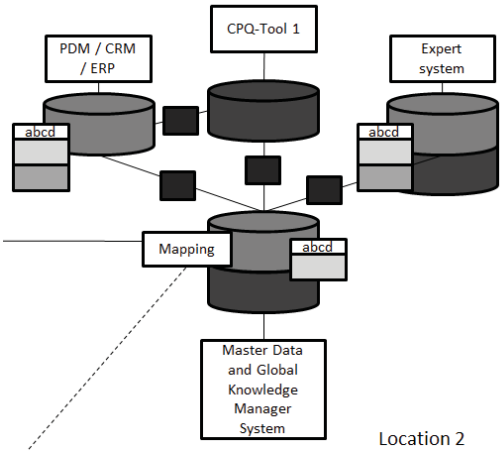


**Figure 8.** CPQ-tool – expert system architecture

These two simplified examples are only supposed to give an overview about the working-principle to be established with the introduction of a GKM and need to be further researched in following publications. But since the vision formulated at the beginning of this paper also includes recommendation mechanisms and possibilities to implement self-learning algorithms the proposed system architecture needs to be accomplished by continuative considerations.

## 6 CONSTRAINT-BASED RECOMMENDER SYSTEMS

Recommender systems are since the 1990s an increasingly used service in mainly e-commerce applications to recommend simple products to users [23]. The user gets recommended products based on previously taken buy decisions (individual information), social background information and a knowledge base that proceeds given user input in form of determined attribute and using concerned domain and contextual knowledge. But according to Felfernigs et al. definitions recommender systems could be used in a much more general way: 'Any system that guides a user in a personalized way to interesting or useful objects in a large space of possible options or that produces such objects as output.' [24].
Besides the possibilities given for e-commerce sellers, a second very large field of application opens with the usage in expert systems. Mainly product configuration applications are adopting recommendation techniques which can be classified in:

1. collaborative recommendation – relying on the choices made previously by other users with the same social and demographical background
2. content-based recommendation – relying on choices made previously by the user herself
3. knowledge based recommendation – relying on user requirements and domain knowledge
4. hybrid forms – try to combine the other types to avoid disadvantages of each stand-alone solution

With respect to the high voltage transmission business the only reasonable choice for recommender systems is the knowledge based recommendation technique since electric power transmission substations do not rely on choices or preferences of single users but on very detailed requirement specifications delivered by the customers (utilities). Even if the requirement specifications of utilities do not change frequently and a content-based recommendation could be possible in terms of parameter input to a configuration application an important circumstance, a unique feature of substations has to be taken into account – the environmental conditions. Unlike other typically configurable products like cars or kitchens substations are crucially impacted by the locations they are assembled.
For example, the Russian ministry of electricity announces in a tender a high voltage transmission substation with the same topology (electrical circuit diagram) and electrical requirements as five years before. One might conclude that the same electrical requirements lead to the same bill of material and same physical topology of the substation. But taking the new location near the Baltic Sea compared to the previous one in Novosibirsk into account this paradigm is false. The environmental conditions including temperature, air pollution, earthquake requirements and altitude of side have an essential impact on the physical arrangement of the substation. This set of conditions to be considered can be depicted as a very complex conglomeration of constraints and is in conclusion applicable for recommendation

systems. However, the complexity of the configuration task is probably too high to be provided with all features in one product configuration application which is why the example in the previous chapter has been chosen.
To achieve the best configuration solution many specialized systems have to give input to the configuration process. Mechanical calculations, waste heat, ferroresonance and further calculation applications are delivering valuable information and need to be addressed to receive a feasible solution. The expert systems outcome is involved as input for the existing recommendation constraints to create a solution for the given configuration task. The user of CPQ-tool 1 in the example illustrated for Figure 7 as a non-expert gets one feasible solution recommended without any repair mechanism. The solution will be rather conservative, but is in line with the given rules and constraints. On the other hand the expert using CPQ-tool 2 in the example has the possibility to neglect the recommended solution and choose a more efficient one. The changes are recognized by the system and adapted to the existing constraints in the GKM. A self-learning system as formulated in the vision at the beginning of this paper is created using expert knowledge to continuously improve the general knowledge manager as a basis for all product configurators. Basis for this mechanism is the system architecture and universally understandable format to provide communication and non-redundancy between all parts of the system as proposed in this paper.

## 7 CONCLUSION AND OUTLOOK

The paper discussed the currently arising challenges and opportunities given in the high voltage transmission business and highlighted the current situation of the information technology in the business. Insulated solutions due to complexity of the products and the world-wide distribution of the associated business segments lead to redundant data storage and the development of several different product configuration solutions. On the basis of the state of the art methods to normalize data and to integrate master data solutions into the system, considerations concerning the resulting system architecture have been made. Furthermore, the need to create a non-redundant knowledge set has been emphasized in addition to a normalized and integrated data set. Deducted from this condition, the proposed system architecture was extended by introducing a global knowledge manager on the same level as the master data system to create a framework for configuration, pricing and quotation rules and constraints. Additionally, a universally understandable format was proposed to enable communication between all parts of the system and to create the possibility to integrate expert systems via inquiries scheduled and processed by the global knowledge manager. The working principle of this system was outlined by two examples. Finally, the opportunity to extend the system by recommendation techniques and self-learning algorithms was pointed out.
Since this paper was supposed to give only a first overview over the problem statement, several future prospects arise from the evaluations presented. These future prospects are the following:

- Data integrity – how can product configurators be enabled to use data from centrally maintained databases instead of using encapsulated exclusive data sets resulting in redundancy?
- Knowledge integrity – how should knowledge normalization methods look like to build the base for a global knowledge manager and a framework for rules and constraints?
- System architecture – which system architecture allows the highest performance, best maintainability, and security with respect to the requirements given by a fully integrated knowledge system?
- Global knowledge manager – how should the global knowledge manager be designed?
- Universally understandable format – how should a format look like which is processible by all databases and programs in the system, including expert systems and product configurators?
- Recommender system – how can rules and constraints be improved by analyzing decisions of experts in a product configurator to improve the recommended solutions to non-experts?

## REFERENCES


[1] European Commission, *Energy - Country datasheets,* https://ec.europa.eu/energy/en/data-analysis/country, (2018)
[2] Katrin Schaber, Florian Steinke, Pascal Mühlich and Thomas Hamacher, *Parametric study of variable renewable energy integration in Europe: Advantages and costs of transmission grid extensions,* Energy Policy, Volume 42, 498 – 508, (2012)
[3] Mathias Uta, *Development of a product configurator for highly modular and modifiable products,* Figure 1, (2017)
[4] Siemens AG, *High-Voltage Products,* https://www.siemens.com/content/dam/internet/siemens-com/global/products-services/energy/high-voltage/high-voltage-switchgear-and-devices, 2016
[5] ABB, High Voltage Products – Business snapshot, http://new.abb.com/high-voltage, (2017)
[6] Graeme C. Simson and Graham C. Witt, *Data Modeling Essentials*, Third Edition, p. 10, (2005)
[7] E. F. Codd, *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM., 13 (6), 377–387, (1970)
[8] Alan Radding, *So what the Hell is ODBMS?*, Computerworld. 29 (45), p. 121–122, (1995)
[9] Frank Manola, *An Evaluation of Object-Oriented DBMS Developments*, GTE Laboratories technical report TR-0263-08-94-165, (1994)
[10] Ramaknth S. Devrakonda, *Object-relational database systems – The road ahead*, Crossroad magazine, Volume 7, p. 15-8, (2001)
[11] Michael Stonebraker, *Object-Relational DBMSs: The Next Great Wave*, Morgan Kaufmann Publishers, p. 12, (1996)
[12] Alejandro Vaisman and Esteban Zimányi, *Data warehouse systems*, Springer, (2014)
[13] ISO 8000-2:2012, *Data Quality*, Part 2: Vocabulary, (2012)
[14] Cornel Loser, Christine Legner and Dimitrios Gizanis, *Master Data Management For Collaborative Service Processes*, International Conference on Service Systems and Service Management, (2004)
[15] Habibullah Jamal and Kiran Sultan, *Performance Analysis of TCP Congestion Control Algorithms*, International Journal of Computers and Communications, Issue 1, Volume 2, (2008)
[16] William H. Inmon, *Building the Data Warehouse*, John Wiley & Sons, p. 31, (1996)
[17] Robert R. Hoffman, *The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology*, AI Magazine, Summer Edition, p. 53-67, (1987)
[18] S. M. Davis, *Future Perfect: Mass customizing*, Addison-Wesley, (1987)

[19] Aparajita Suman, *From knowledge abstraction to management*, Woodhead, p 87-109, (2014)
[20] Alexander Felfernig, Lothar Hotz, Claire Bagley and Juha Tiihonen, *Knowledge-Based Configuration – From Research to Business Cases*, Elsevier Inc., Chapter 7 and 12, (2014)
[21] Frank Manola and Michael L. Broadie, *On Knowledge-Base System Architectures*, On Knowledge Base Management Systems, Springer, p. 35-54, (1986)
[22] Henning Grempel and Gerald Kopatsch, *Schaltanlagen Handbuch*, Cornelsen, 11th Edition, p. 603, (2008)
[23] Michael D. Ekstrand, John T. Riedl and Joseph A. Konstan, *Collaborative filtering recommender systems*, Foundations and Trends in Human-Computer Interaction, Vol. 4, p.81-173, (2011)
[24] A. Felfernig and R. Burke, *Constraint-based recommender systems: technologies and research issues*, In Proceedings of the 10th International Conference of Persuasive Technologies, ICEC '08, ACM, p.1, (2008)

# Configuration Lifecycle Management – An Assessment of the Benefits Based on Maturity

**Anna Myrodia**[1] and **Thomas Randrup** and **Lars Hvam**

**Abstract.** To handle the increasing product complexity manufacturing companies of configurable products tend to utilize configurators to cover more lifecycle phases of their products. This is described as configuration lifecycle management (CLM) and it is concerned with the management of all configuration models across a product's lifecycle. However, to connect and align all configurators and IT systems to each other remains a challenging task. Apart from the technical perspective, on an operational level the integration and alignment of the IT systems also requires a structured approach and is highly related to the maturity of the organization. Therefore, this research focuses on studying the relation between the maturity level and the expected benefits from implementing CLM. It is expected that the more advanced an organization is in using product configurators in different lifecycle phases and integrating and aligning them to each other and to other IT systems, the realized benefits would be significantly higher than the sum of benefits from applying standalone configurators to support each life cycle phase. Empirical evidence from seven case studies demonstrate that there is a relation between the maturity and the realized benefits with regards to the utilization of product configurators.

**Keywords:** configuration lifecycle management, maturity, benefits, product configuration

## 1 INTRODUCTION

Configuration lifecycle management (CLM) describes the management of all configuration models and related data across all lifecycle phases of a product [1]. A CLM solution is highly relevant for manufacturing companies of configurable products, as its purpose is to provide one valid source of configuration data and models that is shared among different business units within an organization.

The utilization of product configurators comes along with various benefits. During the last decades, several researchers have performed studies to identify and measure the realized benefits of the use of a product configurator [2–4]. The identified benefits cover a wide range of aspects, from process improvements to impact on products' profitability. However, the majority of these studies are concerned with configurators that are implemented in the sales phase and some in the engineering phase [2,3] .

Therefore, the focus of this research is to identify possible gains when the utilization of a product configurator is not limited only to the sales phases, but it includes all lifecycle phases of a configurable product, such as engineering, sales, manufacturing and service. It is expected that the realized benefits would be similar but not identical in the remaining lifecycle phases and that the accumulated impact would be significantly higher than the gains on each individual lifecycle phase.

For an organization, to be able to implement and connect product configurators across all lifecycle phases and business processes is considered a rather challenging task. When it comes to the utilization of a configurator in the sales phase, there are numerous challenges identified not only by the literature but also from industrial user cases [5,6]. Resistance to change, difficulties in data acquisition and verification, valid product modeling and maintenance of the models, accurate documentation are some of the most commonly reported challenges in the utilization of product configurators in the sales phases [7,8] .

It could be assumed that similar challenges are expected to be experienced in the other lifecycle phases during the implementation and utilization of a product configurator. However, this research claims that even though some of the challenges would be faced in all lifecycle phases, there several aspects that are not addressed in them. For instance, developing a universal product model to be used by several configurators across all lifecycle phases, business units, even external organizations (e.g. suppliers, resellers, vendors) requires input from various sources and is highly related to numerous dimensions of the organization [9].

In particular, the integration of product configurators with other IT systems for data exchange, as input and/or output of each configuration step, is considered a rather challenging task, especially when it comes to IT systems that are used by several departments [10]. Apart from the technical challenge of connecting, aligning and integrating IT systems with product configurators, the operational perspective is of high importance and it should not be discarded. At an operational level, the process standardization, resources allocation, knowledge sharing and support, established ways of cross-departmental collaboration are some of the factors that are highly related to the success of utilization a CLM solution [6,7,10,11]. Additionally, on a strategic level a clear mission and vision for CLM deployment, communication to all stakeholders and engagement with specific goals for each involved department are of great importance and highly related to the level of success of the CLM solution.

All these aspects mentioned before that influence the success of a CLM solution are related to the maturity of an organization. Maturity in this context does not only describe the development of the IT systems and the possibilities of seamless integration of a universal product model for a CLM solution. Maturity also describes the process and the organizational development, from an operational, strategic and cultural point of view [12,13]. The readiness of an organization to implement and utilize a CLM solution, and the support and involvement of the stakeholders are crucial success factors for a CLM solution.

As a result, it is expected that the more mature an organization is, the higher the realized benefits would be. Therefore, this

[1] Configit A/S, Copenhagen, Denmark, email: amyrodia@configit.com

research relates the expected benefits to the maturity of the organization. The maturity is evaluated in terms of years of implementation of product configurators and the spam of lifecycle phases they cover. The expected benefits of a CLM solution are estimated to be higher than these of standalone configurators in the different lifecycle phases. Exploratory case studies are conducted to examine this proposition.

***Proposition 1*** The size of realized benefits when implementing a CLM solution is related to the maturity of the organization.

The remainder of the paper is structured as follows. Section 2 includes a literature review on the expected benefits from the use of product configurators in different lifecycle phases and the characteristics of maturity of an organization. Section 3 presents the empirical evidence from the case study research and discusses the results. Section 4 provides some overall conclusions regarding the connection of realized benefits and the maturity of an organization when implementing a CLM solution.

# 2 LITERATURE REVIEW

## 2.1 Benefits from implementing product configurators

This section discusses the findings from the literature regarding the expected benefits from implementing and utilizing product configurators. As this field has been examined in detail, we refer to previous work [2,3,11,14–16] and their lists of references. However, to provide an overview we present a short list of realized benefits for the different lifecycle phases (Table 1).

**Table 1.** Benefits per lifecycle phase

| Lifecycle phase | Benefits |
|---|---|
| Sales | Reduction in quotation time |
| | Improve quotation accuracy |
| | Improve control of product portfolio |
| Engineering | Reduction in number of errors |
| | Improve quality of specification and bills-of-materials (BOMs) |
| Manufacturing | Improve quality of production specifications |
| | Improve communication with suppliers |
| | Reduced production costs |
| Service | Reduced installation and maintenance time |
| | Improved predictability in maintenance of products sold |

The benefits are grouped under each lifecycle phase to provide a better overview when it comes to implementing a CLM solution, and they address three main factors: time, quality and cost [17]. However, it should be mentioned that that are some common benefits reported across all lifecycle phases, such as improved process efficiency, reduction of hours spent due to iterations, improved data validity, improved quality due to reduction in the number of errors.

## 2.2 Maturity

The maturity assessment of an organization includes several dimensions and maturity models are the tools used to perform the evaluation. Strategy, processes, IT, organizational structure, knowledge sharing and support activities are among the most widely discussed dimensions in the literature that describe most accurately all functions of an organization [12,18]. The maturity is measured in each of these dimensions; however, the maturity level does not have necessarily to be the same across all of them. This could explain why companies implementing state of the art configurators are still not able to experience all the expected benefits. This is aligned to the findings of [19] that business processes and IT alignment should fit into the organization.

The improvement of configuration management policies and tools, and the establishment of requirement engineering processes are considered top priorities of organization maturity. Seamless integration, knowledge management, monitoring, support and training activities for the users are additional aspects related to the maturity and affect the success of implementing a configuration solution [20]. Empirical studies also indicate that the maturity of IT processes is connected to the gap between organizational targets and processes' aims [21].

Challenges in realizing expected benefits are identified in the sales and planning process [22–24] and are connected to the need of horizontal reorganizational of the structure to include customer and supply chain stakeholders [25]. The current vertical organization structure is a source of delays, increased costs and challenges of managing subcontractors [20]. This is also supported by [26] who claim that when the manufacturing company is in control of the entire supply chain and it is able to coordinate internal and external processes, then it is more mature and can gain a competitive advantage [27,28].

One aspect of knowledge management related to the maturity of an organization is the lack of overview of the product portfolio, which is due to increased complexity. Keeping external variety high to satisfy personalized customer needs to be induced by controlled internal variety and product standardization to avoid increasing costs and complexity [28].

According to [12], the maturity of an organization is increasing based on the level of standardization. That includes both standardization of products and processes. Consequently, this would have direct impact of the realized benefits by utilizing a product configurator, even more when it comes to CLM. However, this alignment and standardization is a task that requires time as it comes along with numerous changes in the organization [6,29]. It is expected that the higher the maturity of an organization is, the higher the gains form the realized benefits would be by the use of product configurators, especially across all lifecycle phases. This is identified as an area not explored by the existing literature.

Even though the research from [12] focuses on the ETO companies, the underlying principles can be extrapolated and used for manufacturers of standard but complex products too, such as the examined case studies. Therefore, this research aims at contributing to this field by providing some empirical evidence to test the developed proposition.

# 3 EMPIRICAL EVIDENCE

To examine the suggested proposition, case research is selected as the research method. The main reason for selecting case research is that allows for comparison of the results across different case companies, where the analysis has been conducted under the same settings and followed a research protocol. In this study, 7 companies are used as cases. Through the case research the under

examination phenomenon is studied in its natural settings and it allows for deeper understanding of phenomena that are not fully examined [30–32]. In this research, the under investigation phenomenon is the one described in the proposition; the relation between the size of realized benefits and the maturity of an organization with regards to the implementation of CLM. The following section provides an introduction to the companies and the set-up of the research, presents and
analyzes the results.

## 3.1 Background

For this study 7 manufacturing companies (A – G) were contacted. All of them are designing, selling, producing and servicing highly engineered and complex products. All the companies have been utilizing product configurators to support at least one lifecycle phase of their products. Furthermore, all 7 companies are large organizations, employing more than 1000 people, and they are operating globally, in terms of market, production facilities and suppliers. They have been utilizing product configurators for at least 2 years before the research was conducted. Table 2 provides an overview of the selected cases regarding their main characteristics and the lifecycle phases they are utilizing a product configurator.

**Table 2.** Overview of the case studies

| Case company | Industrial sector | Lifecycle phase | No. of years utilizing product configurators |
|---|---|---|---|
| A | IE&M (Mechanical) | Sales | 3 |
| B | IE&M (Mechanical) | Sales | 2 |
| C | IE&M (Medical) | Sales | 5 |
| D | IE&M (Mechanical) | Sales | 6 |
| E | Automotive | Sales, Engineering | 7 |
| F | IE&M (Agriculture) | Sales, Engineering | 7 |
| G | IE&M (Electrical) | Sales, Engineering | 3 |

## 3.2 Results

In each of the case companies' data collection included interviews with managers and head of departments that have been using a product configurator. The form of the interviews was semi-structured, to ensure that the relevant data were collected and to allow for some discussions regarding future directions and initiatives towards a CLM solution. All managers were asked the same set of questions to provide information regarding the use of configurators, the lifecycle phases they cover, and the realized benefits they have been experiencing or measuring. The benefits were predefined, based on the results of the literature review. To ensure the validity of the results, two persons from each company we interviewed separately.

During the interviews, the different maturity dimensions were discussed. Since this is an exploratory study, the focus was given on process standardization and cross-organizational collaboration. Process standardization is assessed based on the following two criteria; the number of manual tasks that need to be performed on top of the use of the product configurator, and the generated documentation following the actual configuration process. Cross-organizational collaboration is assessed based on the number of teams from different departments that are using the product configurator or providing input when setting up the configuration models. In addition to these findings, the research team took into account the number of years that each company has been using configurators and the number lifecycle phase they cover, to assess the maturity of each case company. The assigned maturity level varies among low-medium-high. Table 3 presents the results of the analysis.

As it can be seen from Table 3, the number of realized benefits is increasing along with the maturity of the organization. In detail, case companies E, F and G are ranked with medium maturity level due to the fact that they have cross-organizational implementation of product configurators. Even though case company G has been using product configurators for 3 years, which is relatively lower than cases C and D, its level of maturity is still considered to be medium, due to the fact that it has fully standardized and automated processes, and minimum manual work required on top of the use of the configurators across the sales and the engineering teams. In all these three cases, when setting up the product models in the configurator teams from both the sales and the engineering departments were involved. Teams from these two departments also undertake the maintenance and the update of product related data in the configurator, while at the same time product related data for the sales and the engineering phases are handled via the configurator. The realized benefits reported are related to the process standardization, control of complexity, knowledge management and data validity.

Case companies A, B, C and D are utilizing a configurator in the sales phase, therefore the reported benefits are related to cost estimation, quotation and sales efficiency. It should also be mentioned that case company C was the only one able to provide quantitative data regarding the realized benefits. Company C reported that it has managed to reduce the hours used for preparing quotations by 50% (from days to hours). Due to the reduction of errors in the specifications in the sales phase, they have managed to reduce the costs of poor quality in production with 80% due to more accurate production specification.

By summarizing the results can be concluded that there is a relation to the maturity level of an organization and the size of realized benefits. This confirms the under investigation proposition in this study.

## 3.3 Discussion

The benefits identified in the case studies are aligned to the findings from the literature. On a high level it can be concluded that all the benefits can be grouped under the three categories suggested in the literature; time, quality and cost [17]. This conclusion can be used for assigning key performance indicators (KPIs) to monitor and measure the performance of different factors that have a direct impact on these three categories. The KPIs should both cover the lifecycle management aspects of the configurable products and the configuration process itself (detailed examples of KPIs can be found at [33]). By providing quantitative

data the companies would have a more accurate assessment of the improvements they have established due to the use of the product configuration.

Furthermore, the results from the case studies indicate that process standardization is a cornerstone for a successful implementation of configurators. Case company G is such an example; even though the implementation of the configurator s relatively new (3 years) by standardizing the sales and the engineering processes, they managed to achieve the highest number of benefits across the examined cases. This is because by standardizing the processes, the management of configuration models can be improved [34], and the knowledge encapsulated within these models can be used in different lifecycle phase by different users [35]. In the sales phase, the utilization of the configurator is more mature and is usually where the companies are starting. This can be explained by [6] as sales configurators are proven tools and the most popular solutions both in the industry and in academic research.

However, the findings show that several gains can be experienced in the engineering phase. These benefits might be identical to the ones from the sales phase, such as improved efficiency, quality and lead time, but are also phase specific, such as scalability of product models, product platform design and BOM validation.

Nevertheless, the results cannot be generalized to all lifecycle phases based on this case study, as none of there were no empirical evidence from the manufacturing and service phase in these cases. It can be argued, that in a similar way as in the sales and engineering phase, benefits can be gained across all lifecycle phase of a configurable product. It can also be assumed that the more phases the configurators cover, the higher the degree of process standardization and knowledge sharing across the organization.

## 4 CONCLUSION

The scope of this study is to examine the relationship between the realized benefits from the use of product configurators across all lifecycle phase of a product and the maturity level of the organization. The developed proposition is tested in 7 case companies and the study reveals a direct relation between these two variables.

This is an exploratory study. The main limitation of this research is the generalizability of the results, which can be improved by having a more in depth investigation of the phenomenon.

Future research will include more cases that are using product configurators in the manufacturing and service phase. This will be examined in relation to the maturity of the organization, not only in terms of product and process standardization, but also strategic initiatives, knowledge sharing and support, degree of integration of IT systems. Finally, another factor that should be examined is the complexity of the configuration process, regarding the size of the models, the number of features, rules, and the number of users. This could also provide some insight regarding the implementation strategy that would improve the user-friendliness and the acceptance rate of the new system by its users.

## REFERENCES


[1] Configit A/S, CLM DECLARATION, (2015) 1–2. https://configit.com/configit_wordpress/wp-content/uploads/2015/10/CLM-Declaration-2015.pdf (accessed January 8, 2018).

[2] A. Myrodia, K. Kristjansdottir, L. Hvam, Impact of product configuration systems on product profitability and costing accuracy, Comput. Ind. Ind. 88 (2017) 12–18. doi:10.1016/j.compind.2017.03.001.

[3] K. Kristjansdottir, S. Shafiee, L. Hvam, M. Bonev, A. Myrodia, Return on investment from the use of product configuration systems – A case study, Comput. Ind. 100 (2018) 57–69. doi:10.1016/j.compind.2018.04.003.

[4] A. Haug, L. Hvam, N.H. Mortensen, The impact of product configurators on lead times in engineering-oriented companies, Artif. Intell. Eng. Des. Anal. Manuf. 25 (2011) 197–206. doi:10.1017/S0890060410000636.

[5] T. Blecker, N. Abdelkafi, G. Kreutler, G. Friedrich, Product configuration systems: state of the art, conceptualization and extensions, in: Proc. Eight Maghrebian Conf. Softw. Eng. (MCSEAI 2004), 2004: pp. 25–36.

[6] C. Forza, F. Salvador, Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems, Int. J. Prod. Econ. 76 (2002) 87–98. doi:10.1016/S0925-5273(01)00157-8.

[7] M. Heiskala, J. Tihonen, K.-S. Paloheimo, T. Soininen, Mass Customization with Configurable Products and Configurators: A review of benefits and challenges, in: Mass Cust. Pers. Commun. Environ. Integr. Hum. Factors, IGI Global, 2009: pp. 75–106. doi:10.4018/978-1-60566-260-2.ch006.

[8] M. Heiskala, K.-S. Paloheimo, J. Tiihonen, Mass Customisation of Services: Benefits and Challenges of Configurable Services, Tampere, Finland, 2005..

[9] G. Stevens, Integrating the supply chain, Int. J. Phys. Distrib. Mater. Manag. 19 (1989) 3–8.

[10] C. Forza, F. Salvador, Product information management for mass customization: connecting customer, front-office and back-office for fast and efficient customization, Palgrave Macmillan, New York, 2007.

[11] C. Forza, F. Salvador, Product configuration and inter-firm co-ordination: an innovative solution from a small manufacturing enterprise, Comput. Ind. 49 (2002) 37–41.

[12] O. Willner, J. Gosling, P. Schönsleben, Establishing a maturity model for design automation in sales-delivery processes of ETO products, Comput. Ind. 82 (2016) 57–68. doi:10.1016/j.compind.2016.05.003.

[13] R. Batenburg, R.W. Helms, J. Versendaal, The maturity of product lifecycle management in Dutch organizations: A strategic alignment perspective, Prod. Lifecycle Manag. Emerg. Solut. Challenges Glob. Networked Enterp. (2005) 436–450.

[14] A. Myrodia, K. Kristjansdottir, S. Shafiee, L. Hvam, Product configuration system and its impact on product's life cycle complexity, in: IEEE Int. Conf. Ind. Eng. Eng. Manag., 2016. doi:10.1109/IEEM.2016.7797960.

[15] L.L. Zhang, Product configuration: a review of the state-of-the-art and future research, Int. J. Prod. Res. 52 (2014) 6381–6398. doi:10.1080/00207543.2014.942012.

[16] L. Hvam, M. Malis, B. Hansen, J. Riis, Reengineering of the quotation process: application of knowledge based systems, Bus. Process Manag. J. 10 (2004) 200–213. doi:10.1108/14637150410530262.

[17] M.M. Ahmad, N. Dhafr, Establishing and improving manufacturing performance measures, Robot. Comput. Integr. Manuf. 18 (2002) 171–176.

[18] M. Niknam, P. Bonnal, J. Ovtcharova, Configuration management maturity in scientific facilities, Int. J. Adv. Robot. Syst. 10 (2013) 1–14. doi:10.5772/56853.

[19] R. Batenburg, R.W. Helms, J. Versendaal, PLM roadmap: stepwise PLM implementation based on the concepts of maturity and alignment, Int. J. Prod. Lifecycle Manag. 1 (2006) 333. doi:10.1504/IJPLM.2006.011053.

[20] G. Cugola, L. Lavazza, V. Nart, S. Manca, M.R. Pagone, An experience in setting-up a configuration management environment, in: Proc. Eighth IEEE Int. Work. Softw. Technol. Eng. Pract. Inc. Comput. Aided Softw. Eng., IEEE Comput. Soc, 1997: pp. 251–262. doi:10.1109/STEP.1997.615501.

[21] M.A. Vitoriano Vieira, J.S. Neto, INFORMATION TECHNOLOGY SERVICE MANAGEMENT PROCESSES MATURITY IN THE BRAZILIAN FEDERAL DIRECT ADMINISTRATION, J. Inf. Syst. Technol. Manag. 12 (2015) 663–686.

[22] P. Bower, 12 most common threats to sales and operations planning process, J. Bus. Forecast. 24 (2005) 4–14.

[23] L. Lapide, Sales and operations planning part I: The process, J. Bus. Forecast. 23 (2004) 17–19.

[24] J. Piechule, Implementing a sales and operations planning process at Sartomer company: a grass-roots approach, J. Bus. Forecast. 27 (2008) 13–18.

[25] N. Tuomikangas, R. Kaipia, A coordination framework for sales and operations planning (S&OP)_ Synthesis from the literature, Intern. J. Prod. Econ. 154 (2014) 243–262. doi:10.1016/j.ijpe.2014.04.026.

[26] P.E. Stavrulaki, P.M. Davis, Aligning products with supply chain processes and strategy, Int. J. Logist. Manag. 21 (2010) 127–151. doi:10.1108/95740931080001326.

[27] C. Hicks, T. McGovern, C.F. Earl, A Typology of UK Engineer-to-Order Companies, Int. J. Logist. Res. Appl. 4 (2001) 43–56. doi:10.1080/13675560110038068.

[28] M.H. Mello, J.O. Strandhagen, E. Alfnes, Analyzing the factors affecting coordination in engineer-to-order supply chain, Int. J. Oper. Prod. Manag. J. Manuf. Technol. Manag. Iss Int. J. Oper. & Prod. Manag. 35 (2015) 1005–1031. https://doi.org/10.1108/IJOPM-12-2013-0545.

[29] T. De Bruin, Business Process Management: Theory on Progression and Maturity, Queensland University of Technology, 2009.

[30] J. Meredith, Building operations management theory through case and field research, J. Oper. Manag. 16 (1998) 441–454. doi:10.1016/S0272-6963(98)00023-0.

[31] C. Voss, N. Tsikriktsis, M. Frohlich, Case research in operations management, Int. J. Oper. Prod. Manag. 22 (2002) 198–219. doi:10.1108/01443570210414329.

[32] R.K. Yin, Case study research: design and methods, Sage Publications, Thousand Oaks, 2003.

[33] S. Tornincasa, E. Vezzetti, A. Grimaldi, M. Alemanni, Key performance indicators for PLM benefits evaluation: The Alcatel Alenia Space case study, Comput. Ind. 59 (2008) 833–841. doi:10.1016/J.COMPIND.2008.06.003.

[34] Aberdeen Group, The Configuration Management Benchmark Report, (2007) 27.

[35] D. Monticolo, J. Badin, S. Gomes, E. Bonjour, D. Chamoret, A meta-model for knowledge configuration management to support collaborative engineering, Comput. Ind. 66 (2015) 11–20. doi:10.1016/j.compind.2014.08.001.

**Table 3.** Realized benefits per case company

| Benefits | Company | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|---|
| | Maturity (L=Low, M=Medium) | L | L | L | L | M | M | M |
| Sales | Improve quality - Reduction of number of errors | | | X | X | X | X | X |
| | Improved technology management | | | | | | X | X |
| | Increase productivity | X | | | | | X | X |
| | Increased sales | | | | | | | X |
| | Improve competitiveness | | | | | | | X |
| | Reduction in printing costs and distribution of catalogues | | | | | | | X |
| | Improve process efficiency | X | | | | | X | |
| | Reduce cost of IT systems and maintenance | | | | | X | X | X |
| | Improve functionality of integrated IT systems | X | | | | | | |
| | Reduction of complexity | | | | | | X | |
| | Reduced quotation time | X | X | X | X | | | X |
| | Improve accuracy of quotation | X | | X | | | | |
| | Support different market/regions/language/currencies | | X | | | X | | |
| | Improve guided-selling | | X | | | | X | |
| | Increased customer orders | | | | X | | | |
| | Improved dealer management | | | | | | X | |
| | Increase number of quotes through dealers | | | | | | | |
| | Improved ordering process and customer self-service | | | | | | X | X |
| | Improved validity of configuration data | | | | | X | | |
| Engineering | Improve efficiency and scalability of product modeling | | | | | X | | |
| | Bill of material validation | | | | | X | | X |
| | Component optimization | | | | | X | | |
| | Improve quality - Reduction of number of errors | | | | | X | X | X |
| | Improved technology management | | | | | | X | X |
| | Increase productivity | | | | | | X | X |
| | Increased sales | | | | | | | X |
| | Improve competitiveness | | | | | | | X |
| | Reduction in printing costs and distribution of catalogues | | | | | | | X |
| | Improve process efficiency | | | | | | X | |
| | Reduce cost of IT systems and maintenance | | | | | X | X | X |
| | Improve functionality of integrated IT systems | | | | | | | |
| | Reduction of complexity | | | | | | X | |
| | Reduced quotation time | | | | | | | X |
| No. of benefits per case | | 5 | 3 | 3 | 3 | 9 | 15 | 18 |