15th International Configuration Workshop

Proceedings of the 15th International Configuration Workshop

Edited by
Michel Aldanondo and Andreas Falkner

August 29-30, 2013
Vienna, Austria

Organized by
Toulouse University — Mines Albi — CGI, France
Siemens AG, Austria

SIEMENS

MINES Albi-Carmaux

Université de Toulouse

ISBN: 979-10-91526-02-9

École des Mines d'Albi-Carmaux
Campus Jarlard
Route de Teillet
Albi 81013 Cedex 09
France

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

15th International Configuration Workshop

Chairs

Michel Aldanondo, Toulouse University – Mines Albi, France
Andreas Falkner, Siemens AG, Austria

Organizing Committee

Michel Aldanondo, Toulouse University – Mines Albi, France
Andreas Falkner, Siemens AG, Austria
Gerhard Friedrich, AAU Klagenfurt, Austria
Thorsten Krebs, Encoway GmbH Bremen, Germany

Program Committee

Claire Bagley, Oracle Corporation, USA
Conrad Drescher, University of Oxford, UK
Ingo Feinerer, TU Wien Austria
Alexander Felfernig, Technische Universität Graz, Austria
Cipriano Forza, Padova University, Italy
Albert Haag, SAP AG, Germany
Lothar Hotz, Universität Hamburg, Germany
Markus Stumptner, University of South Australia, Australia
Erich Teppan, AAU Klagenfurt, Austria
Juha Tiihonen, Aalto University, Finland
Arnaud Hubaux, University of Namur, Belgium
Élise Vareilles, Toulouse University – Mines Albi, France
Linda Zhang, IESEG business school Paris France

Special Thanks

Paul Gaborit, Toulouse University – Mines Albi, France

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Contents

Foreword 7

Toward automatically learned search heuristics for CSP-encoded configuration problems - results from an initial experimental analysis
Dietmar Jannach 9

Choice Navigation Assessment for Mass Customization
Kjeld Nielsen, Thomas Ditlev Brunoe, Simon Haahr Storbjerg 13

Applications of MaxSAT in Automotive Configuration
Rouven Walter, Christoph Zengler, Wolfgang Küchlin 21

Interactive Configuration of High Performance Renovation of Apartment Buildings by the use of CSP.
Élise Vareilles, Christian Thuesen, Marie Falcon, Michel Aldanondo 29

Configuration Dynamics Verification Using UPPAAL
David Fabian, Radek Mařík 35

Improving configuration and planning optimization: Towards a two tasks approach
Paul Pitiot, Michel Aldanondo, Élise Vareilles, Thierry Coudert, Paul Gaborit 43

Recommender Systems for Configuration Knowledge Engineering
Alexander Felfernig, S. Reiterer, M. Stettinger, F. Reinfrank, M. Jeran, G. Ninaus 51

Solving Object-oriented Configuration Scenarios with ASP
Gottfried Schenner, Andreas Falkner, Anna Ryabokon, Gerhard Friedrich 55

Configuring Domain Knowledge for Natural Language Understanding
Matt Selway, Wolfgang Mayer, Markus Stumptner 63

The effect of sales configurator capabilities on the value perceived by the customer through the customization process
Elisa Perin, Alessio Trentin, Cipriano Forza 71

Generation of predictive configurations for production planning
Tilak Raj Singh, Narayan Rangaraj 79

Choice Navigation: Towards a Methodology for Performance Assessment
Simon Haahr Storbjerg, Kjeld Nielsen, Thomas Ditlev Brunoe 87

What makes the Difference? Basic Characteristics of Configuration
Lothar Hotz 95

(Re-)configuration of Communication Networks in the Context of M2M
Iulia Nica, Franz Wotawa 101

New complex product introduction by means of product configuration
Martin Bonev, Manuel Korell, Lars Hvam 109

Towards Anomaly Explanation in Feature Models
Alexander Felfernig , D. Benavides, J. Galindo, F. Reinfrank 117

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Foreword

Workshop goals

Product configuration is the task of composing product models of complex systems from parameterizable components in the mass-customization business model. This task demands for powerful knowledge-representation formalisms and acquisition methods to capture the great variety and complexity of configurable product models. Furthermore, efficient reasoning methods are required to provide intelligent interactive behavior in configurator software, such as solution search, satisfaction of user preferences, personalization, optimization, diagnosis, etc...

The main goal of the workshop is to promote high-quality research in all technical areas related to configuration. The workshop is of interest for both researchers working in the various fields of applicable AI technologies mentioned below as well as for 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.

Workshop location and history

The Workshop on Configuration 2013 continues the series of successful workshops organized within IJCAI, AAAI, and ECAI since 1999 (for more details, please consult `http://en.wikipedia.org/wiki/Knowledge-based_configuration`). Beside researchers from a variety of different fields, past events also attracted a significant number of industrial participants from major configurator vendors Tacton, SAP, Oracle, Encoway, or IBM-ILOG, as well as from end-users Siemens, Renault, HP, or DaimlerChrysler. In 2013, the workshop is a stand-alone event for the first time and last one and a half days. It takes place in Vienna, Austria at the conference center of Siemens AG Österreich.

The working notes of this workshop gather contributions dealing with various topics closely related with configuration problem modeling and solving. The 16 papers demonstrate both the wide range of applicable techniques and the diversity of the problems and issues that need to be studied and solved to construct and adopt effective configurators.

Michel Aldanondo *and Andreas* Falkner
July 2013

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Toward automatically learned search heuristics for CSP-encoded configuration problems – results from an initial experimental analysis

Dietmar Jannach
Department of Computer Science, TU Dortmund, Germany
dietmar.jannach@tu-dortmund.de

Abstract

Constraint Programming historically been one of the most important approaches for compactly encoding and solving product configuration problems. Solving complex configuration problems efficiently however often requires the usage of domain-specific search heuristics, which have to be explicitly modeled by domain experts and knowledge engineers. Since this is a time-consuming task, our long term research goal is to develop techniques to automatically learn appropriate search heuristics for a given configuration problem.

Compared to other types of Constraint Satisfaction Problems (CSPs), practical configuration problems have certain specific characteristics. First, often only a few of the variables are used to specify the problem ("inputs"); in addition, the specific user inputs and the corresponding final configurations are not equally distributed in the solution space.

In this paper, we present results of an initial simulation-based experimental analysis, in which we aimed to evaluate if already simple statistics can help to speed up the search process. The first results indicate that already trivial branching statistics can help to improve search efficiency.

1 Introduction

Encoding product configuration problems as Constraint Satisfaction Problems (CSPs) [13] has a comparably long tradition both in research and in industrial practice. Using CSPs and Constraint Programming techniques has various advantages, compared, e.g., to rule-based systems, as CSP encodings are declarative in nature and thus often easier to maintain. Furthermore, a number of extensions to the basic CSP encoding scheme as well as specific solving techniques have been proposed in the past, which are at least partially inspired by the specific characteristics of product configuration problems, see [1], [5], [7] or [11]. Today, there also exists a number efficient free and commercial constraint solvers that can be used to check configurations for consistency or to complete partial configurations given some customer inputs.

However, some larger and complex configuration problems can only be solved efficiently when domain-specific heuristics are used that guide the search process. In [4], for example, Fleischanderl et al. report of a configuration problem in the context of telecommunication switches where the final configuration can comprise thousands of interconnected components. The so-called "Partner Units Problem" is another example of a hard real-world configuration problem, for which recently a heuristic algorithm was proposed which allows the problem to be solved in an efficient way [12].

Such heuristics are however domain-specific or even problem-specific and their identification, formalization and evaluation usually is a time-consuming and manual process. It would therefore be desirable to have domain-independent techniques, which help us to automatically derive appropriate heuristics for a given problem setting. In principle, different approaches to achieve this goal are possible. First, one could try to analytically examine the configuration problem (or constraint network) and its solution space. Alternatively, one could follow a learning-based approach by analyzing a number of past solution searches in order to derive appropriate search heuristics.

In our ongoing research, we will focus on the latter type of systems. The long term goal of this research activity being to develop a set of methods that use a learning-based approach to derive search heuristics for configuration problems. We decided to follow the path of a learning-based approach for several reasons. First, in real-world applications, the configuration reasoning process (e.g., to complete a partial configuration) is initiated several times, so that more and more training data will be naturally available over time and the heuristics can thus be made self-adaptive. Furthermore, in many domains, the actual configurations requested by customers are not equally distributed in the solution space and there might be configurations which are far more popular than others[1]. When using a learning approach, such information, which might only be available once the system is deployed, can be integrated in the heuristics learning process.

In this paper, we report the results of an initial experimental analysis, in which we implemented a basic value ordering heuristic for CSPs, which simply ranks the possible variable values based on the number of times they were suc-

[1] In some complex configuration scenarios, every configuration might be unique; still, some subassemblies are usually similar or identical across different configurations.

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

cessfully chosen in previous configuration runs. Our evaluation is based on a simulation, in which we artificially and randomly generated inputs for a number of benchmark problems from a Constraint Solver competition. The corresponding solutions were used as an input for the "training" phase in which statistics were collected. The benchmark problems were then solved again based on this statistics-based heuristic. To compare the efficiency, the required running times were measured. Our initial results show that significant reductions for some types of problems can be achieved even when a very simple learning strategy is applied.

Overall, we consider our work to be first evidence for the general feasibility of such approaches in the configuration domain and as an initial step toward the development of more advanced learning strategies. Our basic technique can furthermore be used as a baseline in further experiments. Finally we propose an experimental evaluation protocol to evaluate the effectiveness of such learning-based approaches.

2 Approach and Initial Results

In our analysis, we focus on standard CSPs which are represented by a tuple $< V, D, C >$, where V is a set of variables, D a set of finite domain associated with these variables, and C a set of constraints on the variables. A solution to a CSP comprises an assignment of values to each problem variable in V such that no constraint from C is violated, see, e.g., [13] for a comprehensive discussion of CSPs.

2.1 The role of branching strategies

When systematic tree search is used as a problem solving scheme for CSPs, the choice of the branching strategy, that is, which variable to consider next and which values to try first, can have a significant impact on the required search time. Over the last decades, a number of different and often domain- or problem-independent branching heuristics have therefore been proposed to speed up the search process.

Consider the following example, which shall demonstrate the impact of the branching strategy on the solution efficiency. When searching for one solution for the classical "all-interval series" problem[2] of a given size without any problem specific optimizations, the running times when using the popular Choco[3] constraint solver with different built-in heuristics range from 30ms to 1 minute. Interestingly, the best strategy for this setting seems to be to pick variable values in decreasing order (30ms), which is an order of magnitude faster than the usual "increasing domain" strategy (500ms) or the dynamic "impact-based branching" [9] strategy (800ms). When no specific strategy is explicitly defined, the search can take up to one minute.

In many cases, the question of which heuristic to chose for a certain problem setting cannot be easily decided analytically and requires an experimental analysis or can be explored with a so-called portfolio solver. Such portfolio solvers, which try out different solvers and corresponding solving strategies based on a case base of past solution searches for similar problem instances, have shown to be very successful in CSP Solver competitions [8].

2.2 Proposed baseline and experimental setup

For our experiments, we extended the open source constraint solver Choco and implemented a new CSP value ordering strategy called `MostFrequent`, which picks the next value to test in the search process simply based on the number of occurrences of this value in solutions in previous search runs[4]. When considering, for example, a PC configurator, if "Intel Core i5" was the most frequent choice in previous configuration sessions, the solver would simply try to use this value first when asked to complete a partial configuration. While in reality the choice of the CPU of course depends on the specific requirements of the current customer, our underlying assumption is that not every possible configuration is equally popular. Therefore, if the specific CPU type was compatible with a larger number of popular and frequent configurations, it might be helpful to try this particular value first (as long as it is not already ruled out by other constraints specified in the current session)[5].

In order to evaluate this approach, we conducted experiments in which we measured the running times when using different branching strategies for a number of benchmark problems of the CPAI'08 solver competition[6]. As a baseline in our comparison the typical built-in `IncreasingDomain` default strategy was used. The chosen benchmark problems used in our experiments, see Section 2.3, had to fulfill certain properties. First, they of course had to be solvable. Furthermore, as we had to run a larger number of solution searches, e.g. to factor out random factors, we picked problems for which the solver could determine a solution (or report infeasibility) for a given set of random inputs relatively quickly, i.e., in less than a second[7]. The experiment consisted of two phases.

(A) Statistics collection phase. Depending on the size of the problem, we randomly designated a small number of the problem variables to be input variables. When the CSP for example contained 50 variables with an average domain size of 20, we, e.g., picked up to 5 variables as inputs, so that we could make sure that several thousand input combinations (and resulting configurations) are possible. Next, we created random input values for these variables, started a solution search and recorded the variable assignments, if a solution was found. In order to simulate that some configurations are more popular than others, we picked the input values using a Gaussian distribution and repeated the process until 300 solutions with the default branching heuristic were found. As we are also interested how the number of training instances effects the statistics-based approach, we defined different measurement points during the simulation run, in which we made

[2] The goal is to find permutations of a given list of numbers that fulfills certain properties, see `http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob007/refs.html`

[3] `http://www.emn.fr/z-info/choco-solver/`

[4] Technically, we used Choco's built-in extension mechanism and implemented a new `ValIterator` class.

[5] Note that the general solution space for a given configuration problem is not affected by the different heuristics.

[6] `http://www.cril.univ-artois.fr/CPAI08/`

[7] We furthermore excluded benchmark problems which could not be imported by Choco's current XML import program.

	Problem name	Default	30	50	100	150	200	300	Diff.
1	normalized-renault-mod-0_ext.xml	143,44	37,03	26,20	26,60	28,72	28,70	30,77	-82%
2	normalized-bibd-8-14-7-4-3_glb.xml	11,71	7,65	6,94	7,71	7,64	8,38	6,48	-41%
3	normalized-squares-9-9.xml	53,75	41,88	44,30	41,72	42,40	43,32	44,90	-22%
4	normalized-geo50-20-d4-75-29_ext.xml (less inputs)	431,25	353,07	366,06	327,03	354,16	342,25	352,29	-24%
5	normalized-geo50-20-d4-75-24_ext.xml	95,11	99,64	91,66	87,19	93,27	100,32	99,39	-8%
6	normalized-costasArray-13.xml	53,36	48,33	47,70	47,18	47,13	46,96	46,49	-12%
7	normalized-air05.xml	856,51	847,22	859,04	850,81	854,80	848,71	853,36	-1%
8	normalized-magicSquare-5_glb.xml (GAUSSIAN)	112,59	142,86	149,65	145,17	140,57	147,66	145,05	25%
9	normalized-magicSquare-5_glb.xml (RANDOM)	122,81	154,49	144,64	137,90	141,06	131,88	139,49	7%

Figure 1: Measurements for example problems. Running times are given in milliseconds.

a snapshot of the collected statistics so far. These snapshots were taken after 30, 50, 100, 150, 200 and 300 solutions. As a baseline for the required running times using the default strategy, we calculated the average search time for the 300 solutions.

(B) Measuring the effects. After the training phase, we repeated the experiment with random inputs 300 times. This time we however used the statistics-based value selection strategy and using the training data for each of the snapshots to analyze the effect when different amounts of training data was available. For cases when individual values never appeared in a previous solution, we used the `IncreasingDomain` strategy as a fallback.

Note that we used the same set of input variables in that phase as in the training phase, but did *not* use exactly the same input values. Instead, we again generated them randomly. Otherwise, a simple solution caching technique would have obviously led to the best results.

Since the total time of finding a solution for many problems depends on the specific set of variables used as an input set, we repeated the whole above-described procedure for 5 times, each time with a different set of input variables. Each problem therefore had to be solved successfully $(300 + 300) * 5 = 3000$ times, which explains why we only considered problems which could be solved efficiently.

2.3 Initial results

As a performance indicator, we used the average CPU time needed for the solver to find a solution. We also collected statistics about the standard deviation of the different runs. Figure 1 shows the average running times for 300 runs using the default strategy and the running times for the `MostFrequent` strategy at different training levels.

The first row shows the results of a benchmark car configuration problem from Renault, which in our view is therefore the most relevant of all measurements. The problem has 111 variables and an average domain size of around 5. As inputs, we used 6 variables, which leads to a range of $5^6 = 15,625$ input combinations. The number of possible configurations is actually lower as not all input combinations correspond to feasible product variants. In this case, about one third (about 5,000) of the randomly generated input combinations were feasible.

Using the solver's built-in default value selection heuristic, the problem could on average be solved in 134.44 milliseconds. When using the statistics-based strategy, however, the average running times could be reduced to 26.20ms, which is a reduction of over 80%. Interestingly, this effect could be achieved already after a very small number of training runs. Later on, when more training data is available, the values seem to slightly increase again. As we did not measure if these differences are statistically different so far, the slight increase could however be a random effect.

When using the default branching strategy, the standard deviation for all experiment runs for the Renault problem was around 220ms. With the statistics-based strategy, this value could be reduced to the half of that. However, when compared to the absolute overall running times, the standard deviation is much higher for the statistics-based strategy. This in general means that some problems can be very efficiently solved with the statistics-based approach, while in some cases the statistics-based strategy can also lead the solver to wrong areas of the search space[8]. Overall, however, the average number of required backtracks, which we measured but do not report here for space reasons, could also be reduced to a third using the statistics-based strategy.

The other problems of our analysis shown in Figure 1 have different characteristics and are in particular not configuration problems but other types of general constraint problems. Problem 2 in Figure 1, for example, is an artificially created one and has over 500 variables and 400 constraints. Also in this case, a significant reduction of the running times could be observed. Problems 4 and 5 are quite similar, but in the case of problem 5 we used many more variables as inputs which led to a drastically higher number of possible inputs while at the same time the solution search was more constrained. As a result, the improvements for Problem 5 were very small. For Problem 7, no improvement was observable. Problems 8 and 9 are classical magic square problems with a highly symmetric problem structure that does not correspond to the typical characteristics of configuration problems. In particular for case 9, in which the inputs were chosen from an equal

[8] In all experiments we limited the allowed computation time to avoid effects of extreme outliers. In the Renault example, the time limit was set to 1,000ms. Situations in which the time frame was not sufficient were however very rare.

distribution, nearly no improvement was achieved with the statistics-based strategy, because every variable value has an nearly equal probability to appear in a random solution. For case 8, a slight improvement could be observed because the inputs values were chosen using a Gaussian distribution.

3 Previous and future works

To the best of our knowledge, limited research has been done so far on the automatic derivation of search heuristics in the area of product configuration. There are, however, approaches in the area of general CSPs that include a learning component in the search process. In the context of our work, we are particularly interested in approaches which aim to learn from previous solution searches or similar problem instances (in contrast to works which try to adapt the search strategy within a single search, often referred to as "online learning", or based on multiple restarts).

In [3], for example, the authors propose an "advice generation" framework for value ordering in CSPs which is based on estimating the likelihood of the existence of at least one solution in the area of the search graph to be explored. Such estimates can be obtained by analyzing a simplified and backtrack-free version of the problem, where the hope is that the number of solutions in the simplified version with less constraints correlates with the solution count for the original problem. Overall, while the general goal of their work is similar to ours, the approach in [3] is not based on past solutions but from an analysis of adapted problem instances, which can also induce significant additional computational costs. In our work, we assume that the solver can learn by collecting information from previous search runs for different users.

In the area of Answer Set Programming, Balduccini in [2] presents an approach for learning branching heuristics from past solution instances. Similar to our work, the proposed DORS framework aims to derive a problem-specific *policy* to guide the solving procedure, i.e. which branch of the search graph should be explored first. While there are differences related to the actual search procedures, the general idea of [2] corresponds to the work presented in this paper. Our future work includes an analysis of how the more advanced but still not very complex approach from [2] can be integrated in the CSP solving process.

Finally, the idea of deriving heuristics from past observations in an automated way can be also found in other application areas. In [10], for example, supervised machine learning techniques are used to at least partially automate the construction of heuristics for the NP-complete problem of instruction scheduling on modern processors. While the specific relation to our work is limited, our future work includes the exploration of techniques such as decision tree learning or classification, see e.g., [6], for the generation of branching heuristics for configuration problems.

4 Summary

Problem-specific search heuristics are a key element for the practical success of many configurator applications. Since the manual definition of such heuristics is time-consuming, our research goal is to develop techniques that help us to learn such heuristics automatically from previous solution searches. In this paper, we have reported results of an initial analysis of the general feasibility of such an approach based on a simulation with small examples and a simple statistics-based branching strategy. Our first results indicate that measurable efficiency improvements can be achieved, when the special characteristics of configuration problems are taken into account.

References

[1] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.

[2] M. Balduccini. Learning and using domain-specific heuristics in ASP solvers. *AI Commun.*, 24(2):147–164, 2011.

[3] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34(1):1–38, 1987.

[4] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, 1998.

[5] A. Haselböck. Exploiting interchangeabilities in constraint-satisfaction problems. In *IJCAI'03*, pages 282–289, Chambery, France, 1993.

[6] H. Ingimundardottir and T. P. Runarsson. Supervised learning linear priority dispatch rules for job-shop scheduling. In *Proc. LION 2011*, pages 263–277, Rome, Italy, 2011.

[7] D. Jannach and M. Zanker. Modeling and solving distributed configuration problems: A CSP-based approach. *IEEE TKDE*, 25(3):603–618, 2013.

[8] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proc. AICS 2008*, 2008.

[9] P. Refalo. Impact-based search strategies for constraint programming. In *Proc. CP 2004*, pages 557–571, Toronto, Canada, 2004.

[10] T. Russell, A. M. Malik, M. Chase, and P. van Beek. Learning heuristics for the superblock instruction scheduling problem. *IEEE Trans. on Knowl. and Data Eng.*, 21(10):1489–1502, 2009.

[11] T. Soininen, E. Gelle, and I. Niemelä. A fixpoint definition of dynamic constraint satisfaction. In *Proc. CP'99*, volume 1713, pages 419–433, Alexandria, Virginia, USA, 1999.

[12] E. Teppan, G. Friedrich, and A. A. Falkner. QuickPup: A heuristic backtracking algorithm for the partner units configuration problem. In *Proc. AAAI/IAAI 2012*, Toronto, Canada, 2012.

[13] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

Choice Navigation Assessment for Mass Customization

Kjeld Nielsen[1], Thomas Ditlev Brunoe[1], Simon Hahr Storbjerg[2]
[1]Department of Mechanical and Manufacturing Engineering, Aalborg University, Denmark
[2]Vestas Wind Systems A/S, Aarhus, Denmark
kni@m-tech.aau.dk

Abstract

In mass customization, the capability Choice Navigation which is defined as the ability to support customers in identifying their own solutions while minimizing the burden of choice, is essential to market high variety product portfolios effectively. We argue that there is a need for methods which can assess a company's choice navigation and their capability to develop it. Through literature study and analysis of choice navigation characteristics a number of metrics are described which can be used for assessment. The metrics are evaluated and analyzed to be applied as KPI's to help MC companies prioritize efforts in business improvement.

1 Introduction

In any company it is essential to offer products which match the needs and desires of customers in order to achieve sales and profit. This is the case for mass producers as well as mass customizers; however in mass customization this issue is somewhat more complex than mass production due to a much higher variety and a more complex product structure. As pointed out by Salvador et al., mass customizers need three fundamental capabilities to be successful (figure 1): 1) Solution Space Development – Identifying the attributes along which customer needs diverge, 2) Robust Process Design – Reusing or recombining existing organizational and value chain resources to fulfill a stream of differentiated customer needs and 3) Choice Navigation – Supporting customers in identifying their own solutions while minimizing complexity and the burden of choice [Lyons et al., 2012; Salvador et al., 2009].

In order for companies to be able to establish themselves as mass customizers or for existing mass customizers to improve performance, it is proposed that a set of methods for assessing the three capabilities is developed. In this paper, the focus is solely on the capabilities for Choice Navigation. The research question for this paper is:

What metrics can be used to assess capabilities for choice navigation and how can these be determined?

The research question is addressed by first defining choice navigation, and in overall terms, which areas should be assessed. Then a literature review is conducted to identify existing metrics. These metrics are evaluated in order to evaluate whetherthey are can be applied to assess the choice navigation performance, and a final set of metrics is developed including newly defined metrics.

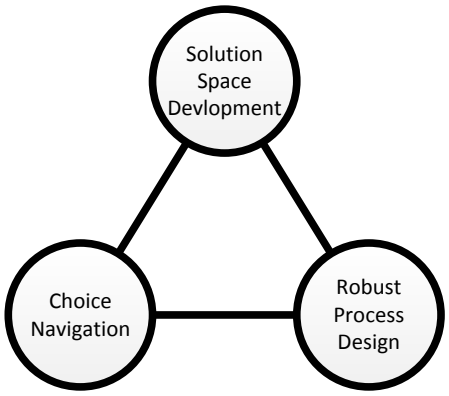


Figure 1 The three fundamental capabilities in mass customization [Salvador et al., 2009]

2 Choice navigation

The capability choice navigation is defined by Salvador et al. [Salvador et al., 2009] as "Support customers in identifying their own solutions while minimizing complexity and the burden of choice". Hence this capability is related primarily to the capabilities of the configuration system, and its ability to configure a variety of products.

Salvador et al. proposes three different approaches to develop the capabilities within choice navigation: Assortment Matching, Fast-cycle, trial-and-error learning and Embedded configuration. However these support the development of choice navigation rather than the assessment of choice navigation capabilities.

Two different perspectives are relevant when assessing a company's choice navigation capabilities. The first perspective addresses the capabilities for supporting the customer in choosing a product which matches the customer's needsfulling. The second perspective is concerned with how well the choice navigation supports the business process involved in product configuration. This paper will focus on the assessment of choice navigation purely from the customer's perspective, thus focusing on the capabilities supporting the customer in the configuration process.

The ideal product configurator should after a customer has finished a configuration leave the customer with the experience that the process has not been unnecessarily difficult to perform and the customer has been able to match his or her needs exactly to a specific configuration of a product [Salvador et al., 2009].

Supporting the customer in the configuration process, thereby making the product configuration task easy and fast, is a matter of aiding the customer in matching characteristics of needs, empowering customers in building models of needs or embedding the configuration in the product itself [Salvador et al., 2009]. Measuring how well choice navigation in a specific company ensures a 100% fit between customer needs and the goods configured by the customers is a somewhat difficult task.

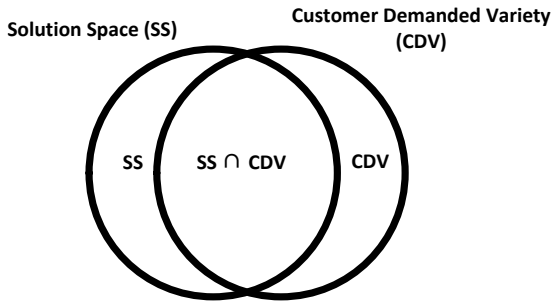


Figure 2 The intersection of offered variety and customer demanded variety yields the potential sellable products.

The problem of assessing the fit between customer needs and a configured product can be described using set theory. Since the objective of choice navigation is to match the customer demand with the offered solution space, a set is defined for each of these as illustrated in figure 2. The optimality of a solution space can then be described by defining two sets of products: 1) the different products offered by an MC company, defined as the set SS (Solution Space) and 2) the variety of products which are demanded by the customers, defined as the set CDV (Customer demanded variety). As illustrated in figure 1, the intersection of the two sets will represent the products offered by the MC company which correspond to products demanded by customers. The intersection of the two sets thus represents the products that customers may buy, given they are able to find and configure the products and willing to pay the required sales price.

Intuitively, maximizing the set SS∩CDV would seem like a good idea since this would maximize the potential number of product variants that can be sold to customers. It would also seem intuitive that the set SS \ CDV i.e. products which are part of the offered variety but are not demanded by customers should be minimized or even eliminated.

When describing these sets, it should be defined which elements are in the set or in other words. What is an element? One possibility would be that each element in the sets corresponds to a unique product variant. Following this, each possible combination of configuration choices would correspond to a variant and thus an element in the set. However, for most MC product families, the number of elements becomes astronomical due to numerous configuration variables each with a number of outcomes. For example, when configuring a Mini Cooper online the configuration choices presented to the customer will result in a number of possible variants well above a 20 digit figure. This is obviously significantly more than the potential market of the Mini Cooper. Assuming that the sale of Mini Coopers is a good representation of the demanded variety, and the Mini Cooper has sold a few million cars and assuming that each sold Mini Cooper is unique, the customer demanded variety will only be a tiny fraction of the offered variety and as a consequence. Furthermore we would expect that assessing whether single variants would counter a demand from a customer is simply not possible if the number of variants is high. Thus it would seem that variants defined as all possible combinations of configuration variables is not an appropriate way to define the solution space set as well as assessing the intersection of SS and CDV.

A more simple and comprehensible way of representing the sets may be defining the elements of the sets as the "dimensions of customization". If a product has a number of customizable attributes and each attribute has a finite number of values that can be chosen, each value will correspond to a product property which can potentially be demanded by a customer.

We thus propose that the solution space is described by the number of customizable attribute's values. For example if a product can be configured in two different sizes and ten different colors, the SS set will contain 12 elements; two size elements and ten color elements. Defining the solution space this way is trivial, since an MC company's offerings will usually be explicit in a configurator, product family model or other documentation. Defining the set CDV on the other hand is far more difficult since it will be impossible or at least extremely time consuming to clarify all potential customers' demand for variety. Also this would depend on the delimitation of the product family's intended customer base. As a result, measuring the size of CDV will expectedly be practically impossible. The intersection of SS and CDV however only describes which products match the demand of customers, and not whether the customers actually buy the products. Whether the customers buy the

products is a matter of several other factors; however the first obstacle is whether the customers are able to match the needs with an actual product configuration, which is the essence of choice navigation. For this reason, we introduce another set, Customer Configuration (CC), which contains the variety that is actually being configured by customers.

The Set CC intersects with both SS and CDV as shown in figure 2, and intuitively the intersection of all three sets SS∩CDV∩CC indicates the optimal situation, where the solution space satisfies a customer demand and the customer is able to configure the product. Conversely, all variety not contained in SS∩CDV∩CC could indicate a problem.

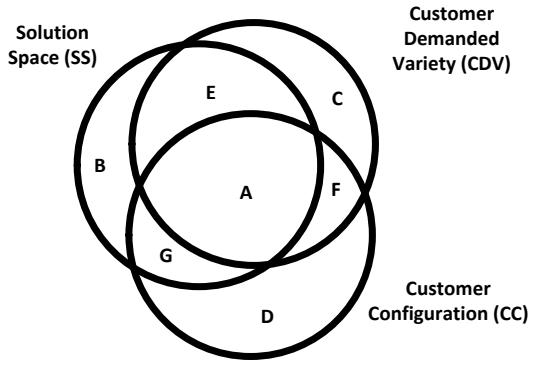


Figure 3 Intersection of Solution Space, Customer Demanded Variety and Customer Configuration

Analyzing figure 3, intersections B and C are consequences of a mismatch between the actual demand and solution space, where B implies variety which is part of the solution space but has no demand thus potentially implying unnecessary complexity costs. C implies a demand for variety that is not met by the current solution space and which may indicate an intersection where the development of the solution space could increase sales. The D intersection is seemingly less interesting in terms of choice navigation, since they relate primarily to the capabilities within solution space development.

In intersection D the customer configures a product that does not meet the demand nor is it contained in the solution space. This is not a typical situation but is nevertheless undesirable, and would likely be indicated by the customer abandoning the configuration. In intersection E, there is a match between the variety offered by the company and the customer demand; however the customer does not configure the product. This is likely a result of a user interface unable to guide the customer satisfactory through the configuration process. Intersection F indicates configuration which match a customer demand, but is outside the actual solution space, i.e. a product that can be configured but not produced, which is also highly undesirable. Finally, in intersection G the customer configures a product that is within the solution space but does not meet the demand thus resulting in a customer disappointment.

The description of the sets CC, CDV and SS above will be used in the following as criteria for evaluating and developing different metrics used for assessing choice navigation capabilities, since metrics indicating variety outside SS∩CDV∩CC will indicate sub optimality within choice navigation.

When assessing a companys capabilities within choice navigation it must be considered within which kind of business environment the configuration will be done. There is typically a great difference in choice navigation setups depending on whether the sales process is done in a business to business (B2B) or in a business to consumer (B2C) sales process. Both setups can be assessed using the same choice navigation metrics, however there are typically differences in the sales setups, where in B2B it is often the sales organization performing the actual configuration process, whereas in B2C this is typically performed by the end customers. Due to this difference, assessessment metrics for choice navigation should be investigated for bias or benchmarking issues when using the results across the different business environments B2B and B2C. We will in this paper not these differences further.

Choice Navigation metrics representing time and effort to reach a configuration, should ideally be developed so that all assessment results could be benchmarked against each other. However regocnising differences between different products and business setups, the metrics should at least allow for benchmarking within a product type and business environment.

One example where differences in product types could make benchmarking between different products non representative is where customers have a great interest in the product and actually wish to spend long time on the configuration process making it more than an experience than a transaction. In this case, a metric indicating high performance for shorter configuration processes might not be representative for the goal the configurator is designed to achieve. Hence, each metric should be scrutinized in relation to assessing a specific product, as special considerations might be relevant for special products.

3 Literature review

Blecker et al. identified and developed a number of metrics for varity steering [Blecker et al., 2003]. Some these metrics are relevant for assessment of choice navigation, and these are identified in the following along with other relevant metrics from literature.

Average configuration length of time metric (CT)

$$CT = \frac{\sum_{i=1}^{N} CT_i}{N} \quad (1)$$

CT: *average configuration lenght of time*
CT_i: *time needed for customer to fulfil one configuration*
N: *number of configurations*

source: [Blecker et al., 2003]

This metric measures how long time a customer or sales person uses for performning the acutal configuration process

Configuration abortion rate metric (CA)

$$CA = \frac{N_a}{N_p} \quad (2)$$

CA: $configuration\ abortion\ rate\ metric$
N_a: $number\ of\ aborted\ configuration\ processes$
N_p: $number\ of\ logins\ (started\ configurations)$

source: [Blecker et al., 2003]

The CA metric describes how frequently customers or sales people choose to abort a configuraiton which has been initiated due to whatever reason.

Customers Return Rate metric (RTR)

$$RTR = \frac{number\ og\ returned\ products}{number\ of\ delivered\ products} \quad (3)$$

source: [Piller, 2002]

The RTR metric describes how often customers returns a product to the company after receiving it due to e.g. disappointment in the product.

Customers Churn Rate metric (CR)

$$CR(\Delta T) = \frac{NOLC(\Delta T)}{NOC(\Delta T) + NONC(\Delta T) - NOLC(\Delta T)} \quad (4)$$

$NOLC$: $number\ of\ lost\ customers\ at\ \Delta T$
NOC: $number\ of\ customers\ at\ T$
$NONC$: $number\ of\ new\ customers\ at\ \Delta T$

source: [Sterne, 2003]

The CR metric describes the relationship between new customers and lost customers.

Customers Repurchase Rate metric (RR)

$$RR = \frac{repurchase\ through\ existing\ customers\ (\Delta T)}{number\ of\ new\ customers\ (\Delta T)} \quad (5)$$

source: [Piller, 2002]

The RR metric describes how often products are repurchased, or how often customers return to byt another different product.

Customers Complaints Rate metric (COR)

$$COR = \frac{number\ of\ complaints\ (\Delta T)}{number\ of\ deliveries\ (\Delta T)} \quad (6)$$

source: [Blecker et al., 2003]

Similar to the CR metric, the COR metric describes how often customers complain over a product they have purchased after receiving it.

Walcher and Piller conducted a survey of 500 different mass customization companies, and for this purpose they developed a number of metrics for comparing the different mass customizers [Walcher & Piller, 2012]. The analysis focused primarily on the configurators, i.e. choice navigation but also on the products. Four objective metrics were included:

- Visual features – To what extent the product is visualized as it is configured, e.g. 2D picture, multuple views, Zoom etc.
- Navigation help – Whether help like progress bars, activity lists, option to save etc. is provided
- Company help – Whether help like recommendations, deeper explanations, design examples etc. is present
- Customer help – Whether users of the configurator is able to get help or inspiration from other users directly or indirectly.

The metrics were evaluated on a scale from 0-4 representing how many of the elements were found in each configurator.
Furthermore, evaluators which were independent mass customization experts were asked to evaluate each configurator using the following subjective metrics:

- Visual realism
- Usability
- Creativity
- Enjoyment
- Uniqueness
- Choice options

Each metric consisted of a number of sub-metrics which the evaluators were asked to assign a rating between 1 and 5. Each configurator was evaluated by 3 different experts and an average was calculated for each metric for each configurator.

4 Choice navigation metrics

In order to evaluate which metrics are usable for evaluating choice navigation capabilities, the different set intersections illustrated in figure 2 are addressed individually. For each intersection, it is evaluated which metrics can support the assessment.

Another requirement for the metrics is that they should be measurable based on readily available data in a company's IT systems, i.e. ERP, CRM, PLM and configuration systems, since this would allow mass customizers to utilize these metrics for continous improvement.

Please note that intersections B and C are disregarded in this context since they relate more to capabilities within solution space development than choice navigation.

4.1 Intersection E

In this case, the customer will start to configure a product, but never reach a final configuration which is purchased, although the solution space supports the requirements. This is difficult to distinguish from the case where requirements cannot be met within the existing solution space (intersection C), however high CA metric can be used as an indication since customers that cannot configure a product to meet their requirements will likely abandon the configuration.

Furthermore, if configurations utilise only a small portion of the solution space and if many configuration variables, rarely deviate from the default values, that may indicate that customers are not aware of all possible variety and have therefore not been able to configure a suitable product although it is in fact offered.

4.2 Intersection F

In this case, customers configure products which are within the customer demanded variety but outside the solution space, i.e. a product is configured which cannot be delivered. This would likely result in the order being cancelled by the company, since it cannot be manufactured. Alternatively, the company will change the configuration to fit within the solution space by e.g. upgrading the product. As an indicator for these configurations we introduce two new metrics:

Seller Order Cancellation rate (SOCR)

$$SOCR = \frac{number\ of\ orders\ cancelled\ by\ seller}{number\ of\ placed\ orders} \quad (7)$$

Seller Order change rate after purchase (SOCRAP)

$$SOCRAP = \frac{number\ of\ orders\ changed\ by\ seller}{number\ of\ placed\ orders} \quad (8)$$

High values of SOCR and SOCRAP would then indicate configurations within intersection F.

Configurations within intersection F as well as D would be a result of a faulty implementation of a configurator, since a configurator should ideally reflect the company's solution space or a subset of the solution space. Reaching configurations within intersection F and D is very undesirable, since it will lead to loss of credibility as well as a need for costly manual business processes to resolve the issue.

4.3 Intersection G

In this case, the customer configures a product which is within solution space but does not correspond to the customer's requirements. In this case several things could happen. If the customer realises that the product is not satisfactory prior to delivery, the customer may cancel the order or change the configuration. To indicate this, two new metrics are introduced:

Customer Order Cancellation rate (COCR)

$$COCR = \frac{number\ of\ orders\ cancelled\ by\ customer}{number\ of\ placed\ orders} \quad (9)$$

Customer Order change rate after purchase (COCRAP)

$$COCRAP = \frac{no.\ of\ orders\ changed\ by\ customer}{number\ of\ placed\ orders} \quad (10)$$

In other cases, customers will not realise that the configured product does not meet requirements, until it is received. In this case the customer may return the product (indicated by RTR) or complain (indicated by COR). Also repurchase rates (RR) and churn rates (CR) would be affected.

Hence configurations within intersection G would be indicated by high values of COCR, COCRAP, RTR and COR and CR and low values of RR.

4.4 Intersection D

In this intersection, the customer configures a product with properties that the customer does not have a demand for and is not part of the solution space. In this case either the customer or the company can react to this and either cancel or change the order. Hence configurations in intersection D will be indicated by High values of SOCR, SOCRAP, COCR and COCRAP. It may however be difficult to determine whether high values of SOCR and SOCRAP are due to configurations in intersection D or F. On the other hand, the customer does not receive the product no matter which are the configuration is in, so whether the customer had a demand for the product may be less important.

4.5 Intersection A

Basically, sales within intersection A are the optimal solution, since products are sold within the solution space which also match the customers' requirements. Hence if there is little indication of configurations outside intersection A, then that should indicate that configurations are within intersection A. Since configurations within intersection A should lead to a sale, then an increase in CSR would also indicate an increase in configurations within intersection A.

Configuration sales rate metric(CSR)

$$CSR = \frac{number\ of\ sold\ configurations}{number\ of\ started\ configurations} \quad (11)$$

4.6 Further metrics

Apart from the metrics which relate directly to the intersections A-G, we identified a number of metrics which may be used to explain why configurations occur in intersections outside intersection A. Hence the metrics can be used to explain the possible reasons for a problem with a configuration system rather than whether there is in fact a problem.

Configuration click index metrics(CI)

$$\mathrm{CI}_c = \frac{\sum_1^n C}{V} \quad (12)$$

CI_c: *clicks index of configuration*
n: *numberofconfiguration in index* (min 100)
C: *number of clicks used for configuration i*
V: *numbers of outcome of variables*

CI metric is a measure of the number of selections, choices or clicks the customer makes in the configurator; or in other words the effort needed by the customer for performing the configuration. It could be the number of selections or actions which the customer has made for a number of given configurations indexed with the total number of variables available in the configurator. The metric cannot be used as benchmark in general or as comparison to other companies/configurators but it can be used internally as an indicator for how a change due to implementation of new variables in the configurator or change of configurator has impacted the choice navigation

performance. Increase of CI may indicate more complex choice navigation or an increase in burden of choice navigation. In a broad view it can be argued that a a value of CI at or near one may indicate a perfect choice navigation.

Time used in configuration index metric(TI)

$$\mathrm{TI}_c = \frac{\sum_1^n T}{V} \quad (13)$$

TI_c: $configuration\ time\ index$
n: $number of configuration\ in\ index\ (min\ 100)$
T: $time\ in\ seconds\ used\ for\ configuration$
V: $numbers\ of\ outcome\ of\ variables$

As for CI the TI metrics gives an index of the time used for a number of given configurations. As for CI the TI may be used internally as an indication of change in burden of choice caused by change of variables and/or change of configurator.

Some of the metrics defined in MC500[Walcher & Piller, 2011] can also be utilized as metrics in this context. However only the objective metrics are included here, and thereby not the metrics that are based on a subjective evaluation. The included metrics are:

- Visual features
- Navigation help
- Company help
- Customer help

All of these metrics are indicators of how customers are guided or helped through the configuration process. Given a company finds that many configurations are observed in intersections E or G, then looking into these metrics may explain the reasons for this.

5 Conclusion & Dicsussion

In order to support the development of choice navigation in mass customization and thereby also product configuration, metrics are needed in order to assess the choice navigation performance. To establish these metrics, relevant literature was reviewed and several applicable metrics were identified. Further metrics were defined in areas where no sufficient metrics could be identified in literature. The following list compiles the metrics identified in literature and newly defined metrics within choice navigation:

Metrics identified in the literature

- Configuration abortion rate metric (CA)
- Customers Return Rate metric (RTR)
- Customers Churn Rate metric (CR)
- Customers Repurchase Rate metric (RR)
- Customers Complaints Rate metric (COR)

Newly defined metrics

- Seller Order Cancellation rate (SOCR)
- Seller Order change rate after purchase (SOCRAP)
- Customer Order Cancellation rate (COCR)
- Customer Order change rate after purchase (COCRAP)
- Configuration sales rate metric(CSR)

It is the intention that these metrics can be used in MC companies for different purposes. One purpose is benchmarking against "best practice" mass customizers, in order to identify areas with the greatest potential for improvement. Another purpose is to use these metrics as key performance indicators which are continually calculated to monitor performance to continuously improve. In relation to research in mass customization it is the intention to apply these metrics in different types of mass customization companies to analyze what distinguishes successful mass customizers.

It is evident that the application of these metrics poses certain requirements related to data availability and quality. However, most MC companies already have systems in place which are very likely to contain the data required for calculating the metrics presented in this paper.

As mentioned in the introduction, choice navigation is one of three fundamental capabilities for successful mass customizers; the other two being robust process design and solution space development. There are strong relations between these three capabilities, and phenomena experienced in a company cannot necessarily be attributed to only one capability, and as such, the metrics defined in this paper can also be influenced by other factors than the solution space development capability.

One example is the metric configuration abortion rate which we argue indicates how well choice navigation is implemented. However, the configuration abortion rate will be strongly influenced by the solution space, i.e. how well the offered variety matches the demanded variety. The value of this metric can thus both be influenced by a company's performance within choice navigation as well as solution space development. In future research, metrics for the other two capabilities, Robust Process Design and Solution Space Development should be established and the links between all three capabilities can be analyzed. Furthermore, the relations between metrics performance and specific methods should be addressed so that an assessment could point out not only what a company should do to improve but also how.

When performing an assessment and interpreting the values of the metrics, the interpretation should take into account the product type. Also when benchmarking, different products cannot necessarily be compared directly. The reason for this is that several metrics are based on the customers actions, and these actions will depend on the product type. For exampe if a customer buys a customized car compared to a customized bag of muesli, then the customer would probably be more likely to complain or return the car if it has a wrong color compared to the muesli, if a wrong ingredient has been added. In that case, the difference would be due to the dfference in cost of the products. Furthermore a metric like the repurchase rate makes more sense for some product types than others. For example, customers are likely to repurchase muesli more often than cars. So this metric would depend on to what extent a product can be characterised as a consumable or a

durable, and in case it is a durable, how long the life cycle is.

With this paper we have ended a preliminary research of assessment and measurement of the mass customization process. We have with this paper finalized a general approach describing how to assess and measure mass customizatioin and developed a framework of potential metrics useful for assessment and measurement of mass customization, whether this is for the purpose of internal performance indicators or it is used for benchmarking in general. Next phase in this research will be test and evaluation of the metrics.

References

[Blecker, T., et al. , 2003]. Key metrics system for variety steering in mass customization. Munich Personal RePEc Archive,

[Lyons, A. C., et al. , 2012]. Mass customisation: A strategy for customer-centric enterprises. Customer-driven supply chains (pp. 71-94)Springer.

[Piller, F. T. , 2002]. Logistische kennzahlen und einflussgroessen zur performance-bewertung der mass-customization-systeme von selve und adidas.

[Salvador, F., et al. , 2009]. Cracking the code of mass customization. MIT Sloan Management Review, 50(3), 70-79.

[Sterne, J. , 2003]. Web metrics: Proven methods for measuring web site successWiley.

[Walcher, D., & Piller, F. , 2012]. The customization 500: A global benchmark study of online BtoC mass customization (1st ed.)www.mc-500.com.

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

Applications of MaxSAT in Automotive Configuration

Rouven Walter and **Christoph Zengler** and **Wolfgang Küchlin***

Abstract

We give an introduction to possible applications of MaxSAT solvers in the area of automotive (re-)configuration. Where a SAT solver merely produces the answer "unsatisfiable" when given an inconsistent set of constraints, a MaxSAT solver computes the maximum subset which can be satisfied. Hence, a MaxSAT solver can compute repair suggestions, e.g. for non-constructible vehicle orders or for inconsistent configuration constraints. We implemented different state-of-the-art MaxSAT algorithms in a uniform setting within a logic framework. We evaluate the different algorithms on (re-)configuration benchmarks generated from problem instances of the automotive industry from our collaboration with German car manufacturer BMW.

1 Introduction

The well-known NP-complete SAT problem of propositional logic—is a given propositional formula satisfiable—has many practical applications; see [Marques-Silva, 2008] for an overview. Küchlin and Sinz [Küchlin and Sinz, 2000] pioneered the application of SAT solving for the verification of the configuration constraints and the bill-of-materials in the product documentation of the automotive industry on the example of Mercedes-Benz. A standard problem to be solved there is the following: Given a (sub-)set $O = \{o_1, \ldots, o_n\}$ of equipment options and a set $C = \{c_1, \ldots, c_m\}$ of configuration constraints whose variables are all options, is it possible to configure a car with the options in O such that C is satisfied? This gives us the SAT problem SAT($C \cup O$), where the options form unit clauses. If the answer is *true*, then the partial configuration O is *valid* and can be extended to a full *valid* configuration F which satisfies C, and F can be readily obtained from the SAT solver.

For the unsatisfiable case, two main questions arise: (1) Which constraints (or clauses for a CNF formula C) of the input formula caused the unsatisfiability? (2) How many (and which) clauses can be maximally satisfied?

*Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, www-sr.informatik.uni-tuebingen.de

The first question can be answered with proof tracing techniques [Zhang and Malik, 2003; Asín *et al.*, 2010]. Here a CDCL SAT solver records a trace while solving the formula. From this trace, a resolution based proof can be deduced, which shows the clauses involved in the unsatisfiable core. An unsatisfiable core is also called conflict.

The answer to the second question can be of important practical use, too. For example, a customer may want to know a maximal *valid* subset of an invalid O. Similarly, the car manufacturer may want to know which maximal subset of C is still satisfied by a currently invalid, but frequently desired option set. This optimization problem can be answered with *MaxSAT*, a generalization of the SAT problem (see Chapter 19 in [Biere *et al.*, 2009]). Instead of deciding the satisfiability of a propositional formula, MaxSAT computes the maximum number of satisfiable clauses in an unsatisfiable formula in CNF. The *Partial MaxSAT* variant splits the clause set into hard and soft clauses in a way that the number of satisfied soft clauses is maximized while all the hard clauses have to be satisfied. In the *weighted* variant of MaxSAT, clauses may carry an additional weight, such as the price of an option o.

Some modern MaxSAT algorithms use SAT solvers as sub-routines by reducing the problem to several SAT solver calls [Fu and Malik, 2006; Marques-Silva and Planes, 2008; Ansótegui *et al.*, 2009]. With this approach, we can make use of all modern techniques (such as clause learning, non-chronological backtracking, or watched literals) of state-of-the-art SAT solvers, which are not generally applicable to MaxSAT solvers.

MaxSAT can be used to answer further questions of practical use. For example: (1) After choosing components with priorities, what is the maximum sum of priorities that can be achieved for a valid configuration? (2) When considering the price of each component, how much is the minimal cost of a valid configuration?

Reconfiguration is of high practical relevance in the automotive industry [Manhart, 2005]. The after-sales business asks for extensions, replacements, or removal of components of a valid configuration with minimal effort. For example, when replacing the alarm system with a newer one, or when moving a vehicle from the U.S. to Europe, we would like to keep the maximal number of already installed components. One approach for reconfiguration uses answer set programming (ASP), which is a decidable fragment of first-order logic

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

[Friedrich *et al.*, 2011]. In this paper, we will describe a MaxSAT based approach for reconfiguration.

This paper is organized as follows. Section 2 defines the MaxSAT variants and notations. In Section 3 we give a short introduction to automotive configuration based on SAT, followed by a complete example. In Section 4 we describe our approach to use MaxSAT for automotive configuration to solve the above questions followed by detailed complete examples. Section 6 shows experimental proof-of-concept results based on different modern MaxSAT solvers. Section 8 concludes the paper.

2 Preliminaries: SAT and MaxSAT variants

A Boolean *assignment* v is a mapping from a set of Boolean variables X to $\{0,1\}$. If a propositional formula φ evaluates to true under an assignment v (denoted as $v \models \varphi$), we call v a *satisfying assignment* or *model* for φ, otherwise an *unsatisfying assignment*. The SAT problem of propositional logic is the question whether such a satisfying assignment v exists for a given formula φ or not.

A literal is a variable or its negation. A clause is a disjunction of literals. Given a propositional formula $\varphi = \bigwedge_{i=1}^{m} \psi_i$ in conjunctive normal form (CNF) over n variables, where ψ_i is a clause for all $1 \leq i \leq m$ and $m \in \mathbb{N}_{\geq 0}$, the solution to the Maximum Satisfiability problem (MaxSAT) is the maximal number of clauses which can be satisfied by an assignment v. Equation (1) shows a formal definition.

$$\mathrm{MaxSAT}(\varphi) := \max \left\{ \sum_{j=1}^{m} \|\psi_i\|_v \middle| v \in \{0,1\}^n \right\} \quad (1)$$

Where $\|\psi_i\|_v = 1$, if $v \models \psi_i$, otherwise $\|\psi_i\| = 0$.

We notice that for the corresponding MinUNSAT problem whose solution is the minimum number of unsatisfied clauses, equation (2) holds.

$$\mathrm{MaxSAT}(\varphi) + \mathrm{MinUNSAT}(\varphi) = m \quad (2)$$

Equation (2) also holds for the same resulting model. As a consequence, we only have to compute one problem to directly get the optimum and the corresponding model for both problems.

There are two extensions of the MaxSAT problem, called Weighted MaxSAT (WMaxSAT) and Partial MaxSAT (PMaxSAT). As the name suggests, in a weighted MaxSAT instance each clause ψ_i has a weight $w_i \in \mathbb{N}_{\geq 0}$ (denoted by the tuple (ψ_i, w_i)). The Weighted MaxSAT problem then asks for the maximal sum of weights of satisfied clauses. Furthermore, in a partial MaxSAT instance, the clauses are divided into disjoint hard and soft clauses sets: $\mathrm{Hard} \dot{\cup} \mathrm{Soft}$. An optimal solution satisfies all hard clauses and a maximal number of soft clauses. Both extensions can be combined to Partial Weighted MaxSAT (PWMaxSAT).

The relationship of equation (2) also holds for each MaxSAT variant.

3 Automotive Configuration with SAT

Automotive configuration can be represented as a constraint satisfaction problem (c. f. [Astesana *et al.*, 2010]) and also as a CNF formula in propositional logic, where each satisfying assignment is called a *valid configuration* of a car. The latter approach was investigated in [Küchlin and Sinz, 2000].

We will give a simplified and short introduction into this representation: (1) Each component (option) c is represented by a separate variable x_c; the component will be used in the final configuration assignment v if and only if $v(x_c) = 1$; (2) components of a family (e.g. different steering wheels) will be restricted by cardinality constraints [Sinz, 2005; Bailleux *et al.*, 2009] to choose exactly one (or at most one, if the component is an optional feature); (3) dependencies between components are expressed as clauses (e.g. the implication $(x_a \wedge x_b) \rightarrow (x_c \vee x_d)$ means "If components a and b are chosen, then component c or d has to be chosen (or both)"; in clause form $(\neg x_a \vee \neg x_b \vee x_c \vee x_d)$).

The resulting formula in CNF is:

$$\varphi_{car} := \varphi_{cc} \wedge \varphi_{dep} \quad (3)$$

Where φ_{cc} are the clauses of the families' cardinality constraints and φ_{dep} are the clauses of the dependencies between the components. With this representation, we can answer the following questions using a SAT solver:

1. **Validation of a partial configuration.**
2. **Forced component:** A component, which is used in every valid configuration.
3. **Redundant component:** A component, which can never be used in any valid configuration.

3.1 Example: SAT based Configuration

We consider the families of components with their limitations listed in Table 1.

Table 1: Component families with limitations

family	alternatives	limit
engine	E_1, E_2, E_3	$= 1$
gearbox	G_1, G_2, G_3	$= 1$
control unit	C_1, C_2, C_3, C_4, C_5	$= 1$
dashboard	D_1, D_2, D_3, D_4	$= 1$
navigation system	N_1, N_2, N_3	≤ 1
air conditioner	AC_1, AC_2, AC_3	≤ 1
alarm system	AS_1, AS_2	≤ 1
radio	R_1, R_2, R_3, R_4, R_5	≤ 1

Furthermore, we consider the dependencies between the components listed in Table 2.

Table 2: Component dependencies

premise	conclusion
G_1	$E_1 \vee E_2$
$N_1 \vee N_2$	D_1
N_3	$D_2 \vee D_3$
$AC_1 \vee AC_3$	$D_1 \vee D_2$
AS_1	$D_2 \vee D_3$
$R_1 \vee R_2 \vee R_5$	$D_1 \vee D_4$

For example, the implication "$G_1 \rightarrow E_1 \vee E_2$" means "If gearbox G_1 is chosen, then engine E_1 or E_2 has to be chosen".

With the resulting formula φ_{car} from the above specifications, we consider two customer cases:

1. A customer chooses engine E_1 and control unit C_1 for the car. But she does not want the air conditioner AC_2. We test Formula (4) for satisfiability.
$$\varphi_{car} \wedge x_{G_1} \wedge x_{C_1} \wedge \neg x_{AC_2} \tag{4}$$
The result is **true**. Derived from the resulting model, we can choose the components D_1, C_1, G_1, E_1 to get a complete valid configuration assignment.
2. A customer chooses the components E_1, G_2, C_2, D_3 and N_2, AC_1, AS_1, R_2. The result is **false**.
The question now is, which maximal subset of the original choice will lead to a valid configuration?

3.2 Advantage of the MaxSAT based approach

With the SAT based configuration, two main problems arise. First, if the configuration is not valid, it is not possible to know which components cause the conflict. Second, even if we know the components causing the conflict, we do not know, which components to omit to get a valid configuration with a maximal number of components we wanted originally. The example 2 of Subsection 3.1 shows such a case.

As mentioned in the introduction, the first problem can be handled with proof tracing to explain a conflict for an invalid configuration. The second problem can be handled with MaxSAT and its extensions. We explain this approach in the next section in detail.

4 Automotive Configuration with MaxSAT

For the representation of automotive configuration as a MaxSAT instance we consider the Partial MaxSAT problem. We use the SAT based specification φ_{car} of Section 3 and divide the clauses into hard and soft ones. First, all cardinality constraints are marked as hard clauses, because they have to be satisfied (e.g. it is not possible to configure a car with more than one steering wheel). Second, it is possible that the dependencies between components do not necessarily have to be satisfied (e.g. a dependency could have been created due to marketing reasons; "No black seats for all Japanese cars"). On the other hand, technical dependencies have to be satisfied (e.g. a conflict between an engine and a gearbox). For simplicity reasons, we also mark all dependencies as hard clauses.

With the representation above, we can consider the following advanced use cases and answer the new arising questions with the help of a Partial (Weighted) MaxSAT solver:

1. **(Maximization of chosen components)** A customer chooses components $c_1, \ldots, c_n$ which lead to an invalid configuration. We can answer the question, what the maximal number of the chosen components for a valid configuration is, by solving Formula (5) with a Partial MaxSAT solver.
$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{x_{c_1} \wedge \ldots \wedge x_{c_n}}_{\text{soft clauses}} \tag{5}$$
2. **(Maximization of priorities)** We can generalize the use case 1 by attaching priorities to the components: A customer chooses components $c_1, \ldots, c_n$ which lead to an invalid configuration. Additionally, the customer has priorities $p_1, \ldots, p_n$, $p_i \in \mathbb{N}_{>0}$, for each component. We can answer the question, which sum of priorities can be maximally reached for a valid configuration by solving Formula (6) with a Partial Weighted MaxSAT solver.
$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{(x_{c_1}, p_1) \wedge \ldots \wedge (x_{c_n}, p_n)}_{\text{soft clauses}} \tag{6}$$
3. **(Reconfiguration)** We can use the introduced techniques in the use cases 1 and 2 for reconfiguration. Let us assume a customer wants to add, replace, or remove components of her existing car. She chooses the components $c_1, \ldots, c_k$ with priorities $p_1, \ldots, p_k \in \mathbb{N}_{>0}$. If the priority or partial state (hard or soft) of a clause of an originally chosen component has changed, the original clause will be replaced by the new partial weighted clause. Otherwise, the clause will be kept. We solve Formula (7) with a Partial Weighted MaxSAT solver to reach the maximal sum of priorities.
$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{(x_{c_1}, p_1) \wedge \ldots \wedge (x_{c_n}, p_n)}_{\text{soft clauses}} \tag{7}$$
To force certain new components to be installed or old components to be kept, we can designate the corresponding clauses as hard clauses.
To reach a valid reconfiguration for the customer, a reconfiguration scenario can be considered as a process in different steps:
   - Check for validation after the customer chooses new components with priorities as previously described.
   - If the hard clauses are unsatisfiable, check for validation after the sales division sets additional dependencies as soft clauses (with priorities).
   - If the hard clauses are unsatisfiable, check for validation after the engineering divison sets additional dependencies as soft clauses (with priorites).

   If the hard clauses are unsatisfiable after all steps, there is no valid configuration, because technical limitations are reached which can not be set as soft clauses. Otherwise, if the hard clauses are satisfiable in one step, we can compute the maximal sum of priorities of the soft clauses while satisfying the hard clauses.
4. **(Minimization of costs)** The components $c_1, \ldots, c_n$ have prices $p_1, \ldots, p_n$, $p_i \in \mathbb{N}_{>0}$. We want to know which components have to be chosen, to get a valid configuration with minimal cost. We can answer the question by solving Formula (8) with a Partial Weighted Min-UNSAT solver.
$$\underbrace{\varphi_{car}}_{\text{hard clauses}} \wedge \underbrace{(\neg x_{c_1}, p_1) \wedge \ldots \wedge (\neg x_{c_n}, p_n)}_{\text{soft clauses}} \tag{8}$$
Instead of finding the minimal costs of a valid configuration, we could also compute a valid configuration of

minimal weights, CO_2 emissions, or other interesting targets.

In all situations above, the resulting model of the solver tells us which components to choose to get the optimum.

Additionally, we can add arbitrary hard clauses to enforce certain constraints: (1) Unit clauses to enforce the in- or exclusion of a component; (2) Additional dependencies between components (e.g. "When engine E_1 is chosen, then choose gearbox G_2"; $(x_{E_1} \rightarrow x_{G_2})$); (3) Additional cardinality constraints (e.g. $x_{D_1} \vee x_{D_2}$ to ensure that one of the dashboards D_1 or D_2 will be chosen).

For example, in Situation 4 (minimization of costs), we could add unit clauses to enforce the inclusion of certain components and then compute the minimal costs of the configuration. The result is a valid configuration with minimal costs which includes our chosen components.

4.1 Example: MaxSAT based Configuration

We reconsider the example in Subsection 3.1.

1. In the second case, the choice of the customer was unsatisfiable. With the MaxSAT based approach of configuration we can find an assignment of a valid configuration where a maximum number of components is included. After solving Formula (5) with a Partial MaxSAT solver, we obtain the results shown in Table 3.

Table 3: Customer choices and Partial MaxSAT results

family	choice	result
engine	E_1	E_1
gearbox	G_2	G_2
control unit	C_2	C_2
dashboard	$\boldsymbol{D_3}$	$\boldsymbol{D_1}$
navigation system	N_2	N_2
air conditioner	AC_1	AC_1
alarm system	$\boldsymbol{AS_1}$	**–**
radio	R_2	R_2

We can reach a valid configuration by changing two of the choices (bold rows in the table) and therefore, we can keep 6 of our 8 original components at most. For the alarm system, the resulting model did not set another alarm system variable to true, because this is an optional feature.

In general, the result obtained from the solver may not be the only optimum. There can be other different assignments with the same number of satisfied clauses.

2. We consider another case, where the customer chooses the components with priorities as shown in Table 4. Additionally, she wants dashboard D_2, D_3, or D_4. To enforce this constraint, we add the hard clause $(x_{D_2} \vee x_{D_3} \vee x_{D_4})$.

Table 4: Customer prioritized choices and PWMaxSAT results

family	choice	priority	result
engine	E_1	8	E_1
gearbox	G_2	5	G_2
control unit	C_2	7	C_2
	$\boldsymbol{D_2}$	**8**	
dashboard	$\boldsymbol{D_3}$	**15**	$\boldsymbol{D_4}$
	$\boldsymbol{D_4}$	**15**	
navigation system	$\boldsymbol{N_2}$	**20**	**–**
air conditioner	$\boldsymbol{AC_1}$	**7**	**–**
alarm system	$\boldsymbol{AS_1}$	**2**	**–**
radio	R_2	15	R_2

Table 4 shows the result, scoring 50 priority points, after solving Formula (6).

3. After the previous configuration, the customer wants to reconfigure her existing car. Table 5 shows her choice. We can imagine that for technical or financial reasons, the engine E_1 and gearbox G_2 can not be replaced. We set them as hard clauses. However, control unit C_2 and dashboard D_4 can possibly be replaced and therefore are set as soft clauses.

Table 5: Reconfiguration choice and PWMaxSAT results

family	state	new priorities	choice	results
engine	E_1	hard	E_1	E_1
gearbox	G_2	hard	G_2	G_2
control unit	C_2	(5, soft)	C_2	C_2
dashboard	$\boldsymbol{D_4}$	**(2, soft)**	$\boldsymbol{D_4}$	$\boldsymbol{D_2}$
navigation system	–	(10, soft)	N_3	N_3
air conditioner	–	hard	$AC_1 \vee AC_2$	AC_2
alarm system	–	(5, soft)	AS_1	AS_1
radio	$\boldsymbol{R_2}$	**(13, soft)**	$\boldsymbol{R_2}$	**–**

The results show that dashboard D_4 was replaced by dashboard D_2 and radio R_2 has to be removed in favor of other components.

4. Now we associate the components with prices (as shown in Table 6) and we want to know a valid configuration with a minimal total price.

Table 6: Components with prices

family	alternatives				
engine	E_1	E_2	E_3		
price (€)	4,000	2,500	4,500		
gearbox	G_1	G_2	G_3		
price (€)	500	800	300		
control unit	C_1	C_2	C_3	C_4	C_5
price (€)	800	2,000	1,500	1,600	1,200
dashboard	D_1	D_2	D_3	D_4	
price (€)	300	500	600	450	
navigation system	N_1	N_2	N_3		
price (€)	100	150	130		
air conditioner	AC_1	AC_2	AC_3		
price (€)	180	100	90		
alarm system	AS_1	AS_2			
price (€)	300	250			
radio	R_1	R_2	R_3	R_4	R_5
price (€)	100	80	200	180	150

For the minimal costs we solve Formula (8) with a Partial Weighted MinUNSAT solver. For the maximal costs we solve Formula (6) with a Partial Weighted MaxSAT solver by considering the prices as priorities. The results are:

- Minimal cost: € 3,900
- Maximal cost: € 8,625

Table 7 lists the components to choose to reach the minimal and maximal costs.

Table 7: Choices for minimal and maximal costs

	choice	
family	minimal cost	maximal cost
engine	E_2	E_3
gearbox	G_3	G_2
control unit	C_1	C_2
dashboard	D_1	D_3
navigation system	–	N_3
air conditioner	–	AC_2
alarm system	–	AS_1
radio	–	R_3

5 Algorithmic techniques

In order to give the reader an impression of how MaxSAT can be computed, we present a short incomplete overview of some algorithmic techniques.

Branch-and-Bound The general branch and bound approach to explore the search tree of optimization problems can also be used for solving MaxSAT and its extensions. Each node of the tree represents a variable of the instance and has two children for the two values the variable can be assigned to. Tree pruning is used as soon as a partial solution becomes worse than the best solution found elsewhere in the tree. Heuristics have been developed e.g. by Wallace and Freuder to narrow the search space predicting the final value of partial solutions [Wallace and Freuder, 1993].

Basic SAT-based Given an unsatisfiable SAT problem $\varphi = \{C_1, \ldots, C_m\}$, we may iteratively try to remove individual clauses C_i until the subproblem φ' becomes satisfiable. φ' will then be maximal in the sense that adding another clause will make it unsatisfiable, but another, larger, subproblem may exist which could be found by removing clauses from φ in a different order.

In SAT solving, clause removal can be simulated by augmenting each clause C_i with a fresh *blocking variable* b_i. As long as b_i is set to *false*, the solver needs to satisfy C_i, but the constraint C_i can effectively be blocked by setting b_i to *true* instead. Now, in order to remove as few clauses as possible, we add m blocking variables to φ as above and restrict the use of the b_i by an additional *cardinality constraint* $CC(k)$, which is a formula that prevents more than k of the b_i to be set to *true*. Iterating over k from below until $\varphi(k)$ becomes satisfiable, or from above until $\varphi(k)$ becomes unsatisfiable, gives us the MaxSAT result $m - k$, and the subset of clauses whose b_i are set to false forms one satisfiable subset of maximum cardinality.

Algorithm 1 reflects basic approach. One improvement of this approach is the use of binary search.

Algorithm 1: Basic SAT-based approach

Input: $\varphi = \{C_1, \ldots, C_m\}$
Output: Minimal number of unsatisfied clauses
$\varphi \leftarrow \{C_1 \vee b_1, \ldots, C_m \vee b_m\}$
$cost \leftarrow m$
while $\mathrm{SAT}(\varphi \cup \mathrm{CNF}(\sum_{i=1}^{m} b_i < cost))$ **do**
  $cost \leftarrow cost - 1$
return *cost*

Core-guided SAT-based Modern proof-tracing SAT solvers return an unsatisfiable subset (unsat core) $\mu \subseteq \varphi$ when given an unsatisfiable φ. It is then clear that at least one clause of μ has to be blocked before φ can become satisfiable, and thus the search can be narrowed compared to the basic approach. An algorithm based on this idea was proposed by Fu and Malik for partial MaxSAT [Fu and Malik, 2006]. In every iteration where the instance is unsatisfiable, we add a new blocking variable to all soft clauses of the unsatisfiable core and a new cardinality constraint to achieve that exactly one of the currently added blocking variables has to be satisfied. We can not just iterate over the unsat cores and count them, because they may not be disjoint.

This idea can also be extended for partial weighted MaxSAT [Ansótegui *et al.*, 2009].

6 Experimental Results

For our benchmarks we used product configuration formulas of a current (2013) product line of the German car manufacturer BMW. We added unit clauses to create unsatisfiable customer orders. We defined the following three categories for hard and soft clauses:

- Order: Soft clauses are unit clauses of the customer's order. All other clauses are hard. This asks, wich of the

customer's wishes can be maximally satisfied.

- Packages: Soft clauses are clauses which represent packages, e.g. a sports package, which triggers all relevant sports components. The unit clauses of the customer's order and all other clauses are hard. This asks, which of the package restrictions can be maximally satisfied w.r.t. the customer's wishes.
- Packages & more: Soft clauses are package clauses and additional other sales relevant conditions. The unit clauses of the customer's order and all other clauses are hard. This asks, which of the package restrictions and additional restrictions can be maximally satisfied w.r.t. the customer's wishes.

The upper half of Table 8 shows detailed statistics about each category. The second half of the table shows how many instances have an optimum. No optimum means that there is at least one conflict involving only hard clauses. The average optimum is the average of the result of the minimal number of unsatisfiable clauses. For example, the average optimum of 2.127 within the 'Order' category means that on average 2.127 of the customer's choices can not be satisfied.

Table 8: Benchmark details

	Benchmark categories		
	Order	Packages	Packages & more
#instances	777	777	777
Avg. #variables	896	896	896
Avg. #hard clauses	4474	3928	3592
Avg. #soft clauses	15	561	897
#no optimum	0	688	0
#with optimum	777	89	777
Avg. optimum	2.127	1.348	4.067

We applied our benchmarks to three different state-of-the-art MaxSAT solvers, namely:

- akmaxsat [Kügel, 2012]: A partial weighted MaxSAT solver based on a branch-and-bound approach. One of the best performing solvers in last year's MaxSAT competition[1].
- Fu & Malik [Fu and Malik, 2006]: A partial MaxSAT solver based on exploiting unsatisfiable cores and adding blocking variables to each soft clause of each found unsatisfiable core.
- PM2 [Ansótegui *et al.*, 2009]: A partial MaxSAT solver based on exploiting unsatisfiable cores. But unlike the Fu & Malik solver this approach only uses exactly one blocking variable to each clause.

For akmaxsat we used the implementation of Adrian Kügel[2]. We implemented the Fu & Malik and PM2 algorithms on top of our own Java SAT solver, which is optimized for our industrial collaborations. The cardinality constraints in the Fu & Malik approach are only of the form $\sum_{i=1}^{n} x_i = 1$ for given variables $x_1, \ldots, x_n$. We encode this constraint through the constraints $(\bigvee_{i=1}^{n} x_i)$ and $\left(\bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\neg x_i \vee \neg x_j)\right)$. The cardinality constraints in the PM2 approach uses general limitations, which we implemented with the encoding proposed in [Bailleux *et al.*, 2009].

All our benchmarks were run on the same environment: Operating System: Ubuntu 12.04 64 Bit; Processor: Intel Core i7-3520M, 2,90 GHz; Main memory: 8 GB; JVM 1.7.0 (for Fu & Malik and PM2).

Table 9 shows the results of our time measurements of each solver in each category. The listed times are the average times a solver needed to solve an instance of a category. We listed the average time in each category Solver akmaxsat has an average time of remarkable less than 0.6 seconds in each category. Our implementation of Fu & Malik has a reasonable average time of less than 6 seconds in each category. Our implementation of PM2 has a reasonable average time for the first category 'Order', but exceeded our time limit of $3,600$ seconds per instance on too many instances of categories 'Packages' and 'Packages & more' to get a reasonable average time.

Table 9: Benchmark results with a time limit of 3,600 sec. per instance

Avg. time (sec)	akmaxsat	Fu & Malik	PM2
Order	**0.165**	4.367	4.180
Packages	**0.025**	1.664	exceeded limit
Packages & more	**0.535**	5.387	exceeded limit

Figures 1, 2 and 3 show the performance of each solver in the first category 'Order'. These figures show the relation between the optimum and the response time of the instances. Especially for Fu & Malik and PM2 the response time seems to grow linearly with increasing optimum.

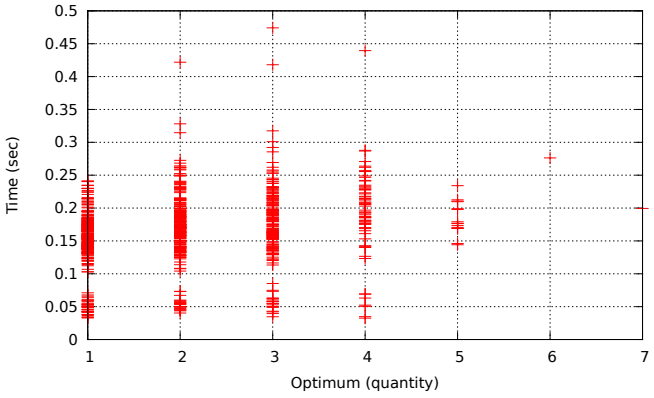


Figure 1: Benchmark 'Order' with akmaxsat

[1] http://maxsat.ia.udl.cat:81/12

[2] http://www.uni-ulm.de/in/theo/m/alumni/kuegel.html

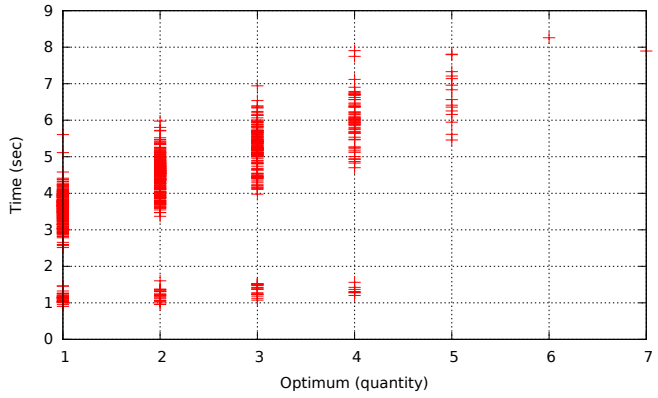


Figure 2: Benchmark 'Order' with Fu & Malik

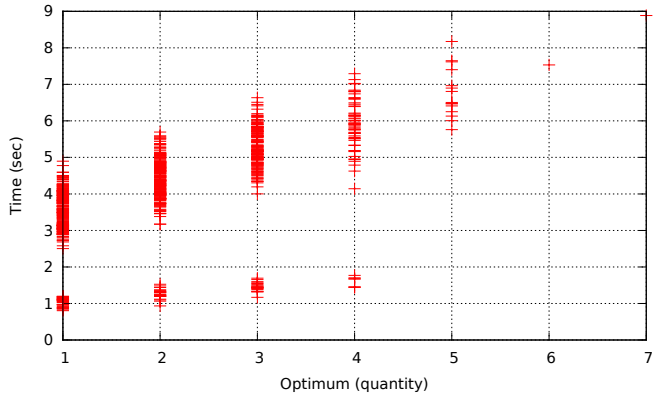


Figure 3: Benchmark 'Order' with PM2

In Figure 4 we can also recognize the linear growing response time with increasing optimum. Also note the lower line of quickly solvable instances.

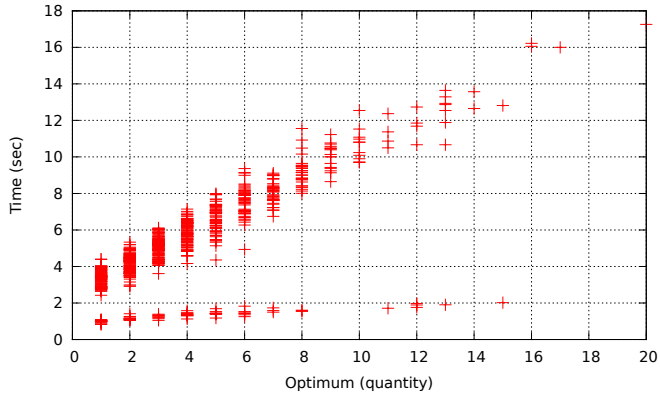


Figure 4: Benchmark 'Packages & more' with Fu & Malik

7 Related Work

In [Junker, 2004] general satisfaction problems are considered, where we have a knowlegde base of constraints which have to be satisfied and customer requirements, which we would like to satisfy. In the context of MaxSAT, the knowledge base can be considered as hard clauses, whereas the customer requirements can be considered as soft clauses. In the case of inconsistency, the proposed algorithm QuickXplain delivers preferred explanations, which are based on a given total ordering of the constraints.

The work of [Reiter, 1987] proposes an algorithm for computing minimal diagnoses using a conflict detection algorithm. A diagnosis is a minimal subset Δ of the customer requirements, such that the constraints without Δ is consistent. In [Felfernig *et al.*, 2012] another algorithm is proposed, called FastDiag, which computes a preferred minimal diagnosis without calculating the corresponding conflicts.

8 Conclusion

In this paper we showed detailed examples of how MaxSAT and its extensions can be applied in automotive configuration. With this approach we are able to repair an unsatisfiable customer order by computing the optimal solution which satisfies as many of the customer's choices as possible. Furthermore, we showed how MaxSAT also can be used in reconfiguration scenarios. From an already configured car we can compute the minimal number of components to change when adding, changing, or removing components.

We created realistic benchmarks for our MaxSAT applications out of the product formulas of our commercial collaboration with BMW. Our time measurements of these benchmarks against the state-of-the-art MaxSAT solvers akmaxsat, Fu & Malik, and PM2, showed that we have a reasonable response time, except for PM2 in two categories. These results suggest that MaxSAT can be applied for industrial automotive (re-)configuration problems.

References

[Ansótegui *et al.*, 2009] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On solving MaxSAT through SAT. In *Proceedings of the 2009 conference on Artificial Intelligence Research and Development*, pages 284–292. IOS Press Amsterdam, Amsterdam, Netherlands, 2009.

[Asín *et al.*, 2010] Robert Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Practical algorithms for unsatisfiability proof and core generation in SAT solvers. *AI Communications*, 23(2–3):145–157, 2010.

[Astesana *et al.*, 2010] Jean Marc Astesana, Yves Bossu, Laurent Cosserat, and Helene Fargier. Constraint-based modeling and exploitation of a vehicle range at Renault's: Requirement analysis and complexity study. In *Proceedings of the 13th Workshop on Configuration*, pages 33–39, 2010.

[Bailleux *et al.*, 2009] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean

constraints into CNF. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing—SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer Berlin Heidelberg, 2009.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, volume 185. IOS Press, Amsterdam, Netherlands, 2009.

[Felfernig *et al.*, 2012] 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, 2012.

[Friedrich *et al.*, 2011] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. (re)configuration using answer set programming. In Kostyantyn Shchekotykhin, Dietmar Jannach, and Markus Zanker, editors, *IJCAI-11 Configuration Workshop Proceedings*, pages 17–24, Barcelona, Spain, July 2011.

[Fu and Malik, 2006] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing—SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer Berlin Heidelberg, 2006.

[Junker, 2004] Ulrich Junker. QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 167 – 172. AAAI Press / The MIT Press, 2004.

[Küchlin and Sinz, 2000] Wolfgang Küchlin and Carsten Sinz. Proving consistency assertions for automotive product data management. *Journal of Automated Reasoning*, 24(1–2):145–163, 2000.

[Kügel, 2012] Adrian Kügel. Improved exact solver for the weighted max-sat problem. In Daniel Le Berre, editor, *POS-10. Pragmatics of SAT*, volume 8 of *EPiC Series*, pages 15–27. EasyChair, 2012.

[Manhart, 2005] Peter Manhart. Reconfiguration – a problem in search of solutions. In Dietmar Jannach and Alexander Felfernig, editors, *IJCAI-05 Configuration Workshop Proceedings*, pages 64–67, Edinburgh, Scotland, July 2005.

[Marques-Silva and Planes, 2008] João Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Test in Europe*, pages 408–413. IEEE, 2008.

[Marques-Silva, 2008] João Marques-Silva. Practical applications of boolean satisfiability. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 74–80. IEEE, 2008.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 – 95, April 1987.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming—CP 2005*, Lecture Notes in Computer Science, pages 827–831. Springer Berlin Heidelberg, 2005.

[Wallace and Freuder, 1993] Richard Wallace and Eugene C. Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *Discrete Mathematics and Theoretical Computer Science*, pages 587–615. American Mathematical Society, USA, 1993.

[Zhang and Malik, 2003] Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the conference on Design, Automation and Test in Europe*, volume 1, pages 10880–10885. IEEE Computer Society, 2003.

Interactive Configuration of High Performance Renovation of Apartment Buildings by the use of CSP

É. Vareilles[a], C. Thuesen[b], M. Falcon[a,c] and M. Aldanondo[a]
[a]: Toulouse University, Mines Albi-Carmaux - France
[b]: Technical University of Denmark - Denmark
[c]: TBC Générateur d'Innovation, Colomiers - France
Corresponding author: elise.vareilles@mines-albi.fr

Abstract

This paper is a prospective study which looks at the possibility of configuring high performance renovation of apartment buildings by the use of constraint satisfaction problem (CSP). This study is one part of a project called *CRIBA* which aims to industrialize high performance thermal renovation of mid-rise (up to seven stories) apartment buildings. The renovation is based on external rectangular panels, always comprising insulation and cladding, and sometimes including, in addition, doors, windows or solar modules. The panels can be fixed directly onto the walls or onto a metal structure around the whole building. With the new thermal envelope and equipment, the building must achieve a really low energy performance of $25\ kWh/m^2/year$. A configuration system, based on CSP approaches, will simultaneously enable the interactive definition of the renovation, the associated bill of material (BOM) and the building site assembly process. In Section two, we set out the industrial problem of residential buildings renovation and explain how a configurator can support it. Then, in the third section, the renovation configuration process is described. In the fourth and final section, we present how the renovation configuration can be addressed with constraints, and we introduce relevant CSP approaches. Through out the article, industrial examples illustrate our proposal.

1 Introduction

The global contribution from buildings (residential and commercial) towards energy consumption has steadily increased. Buildings account for around 20% and 40% of the total final energy consumption in developed countries: 37% in the EU [Perez-Lombard *et al.*, 2008], 36% in the USA [Council, 2013] and 31% in Japan [Center, 2012]. It has now exceeded the other major sectors: industry and transportation. Growth in population, enhancement of building services and comfort levels, together with the rise in time spent inside buildings assure the upward trend in energy demand will continue in the future. Therefore, reducing energy consumption of the building sector is one of our century's challenges. For this reason, it is a prime objective for energy policy at regional, national and international levels.

In several countries, research works are carried out on the efficient measures to take to reduce energy consumption of the building stock. Most states set regulations to improve the energy performance of new buildings. However, the annual rate of construction of new dwellings is only 1.1% in Europe [Poel *et al.*, 2007]. It is therefore very important to renovate the existing buildings to really reduce their energy consumption and to assist the retrofit process by the development of decision support systems [Juan *et al.*, 2010].

This study is one part of a research project called *CRIBA*, which aims to industrialize high performance thermal renovation of apartment buildings. In this project, a very innovative renovation system based on large timber frame panels will be designed. Moreover, all the tools needed to industrialize the renovation process will be developed:

- a new method for three-dimensional building survey and modelling,
- a configuration system for the design of the buildings new thermal envelope (bill of material and assembly process),
- a working site planning model with resource constraints.

The aim of this paper is to present a prospective study on the development of the interactive configuration system for the renovation of apartment buildings.

Therefore, the remainder of the paper is organized as follows. In Section 2, we present the building renovation problem and how the configurator can support apartment buildings renovation. In Section 3, we put forward some ideas on the generic renovation bill of material. In Section 4, we outline the building renovation configuration process. In Section 5, we identify the different kinds of constraints that are needed in order to make the apartment building renovation model.

2 Building Renovation Configuration Needs

In this section, we introduce the building renovation problem which is at the origin of our work. Then, we express what the configurator is expected to generate as results.

2.1 Building Renovation Problem

The industrialized high performance thermal renovation is based on external rectangular panels, always comprising in-

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

sulation and cladding, and sometimes including, in addition, doors, windows or solar modules. Although the shape of the panels is a major limitation for the architectural creativity, this assumption is the key of renovation industrialization and matches most of apartment buildings.

The building sector is very dependent on hand-made methods which are not always synonymous with quality guarantee [Falcon and Fontanili, 2010]. Therefore, the aim of the *CRIBA* project is to prefabricate all the panels needed for a renovation, in a correct order, then to deliver them directly to the working site and finally to hang them on the faades. Therefore, the renovation process enables thermal renovations:

- at low cost, considering all the positive elements, fixed cost, logistic, etc,
- in a short time,
- of high quality,
- in a good environmental balance,
- without rehousing the inhabitants during the renovation works.

Depending on the building strength of materials, the panels can be fixed directly onto the faades or onto a metal skeleton around the whole building. With the new thermal envelope and equipment, the building must achieve a really low energy performance of $25kWh/m^2/year$. In order to reach such a low energy performance, the new thermal envelope has to wrap the whole building. All the faades are covered by non-overlapping panels and are space-partitioned.

2.2 Building Renovation Configuration

The interactive configuration system for the renovation of apartment buildings will simultaneously enable the interactive definition of the renovation thanks to the associated bill of material [Felfernig, 2007] and the building site assembly process.

The bill of material is a list of the components and sub-components, sub-assemblies, and the quantities of each needed to manufacture an end product. It can have multiple options and variants. In our case, we consider:

- the new thermal envelope as the end product;
- the facade new envelopes as the sub-assemblies;
- the complete panels as sub-components;
- the configurable components as leaves of the bill of material (BOM):
  - the panels, which are placed on the faades, include wood structure, insulation and cladding (three or four types at the moment), as shown in Fig. 1,
  - the angles, which make the junction between two faades. An angle is a specific type of panel which cannot include other components,
  - the windows, doors, solar modules and balconies,
  - the metal fasteners, which are used to fix either metal profiles or directly the panels onto the faades. There are several types of metal fastener depending on their type (fasteners to fix metal profiles, to hang panels or to provide wind bracing of panels), their load bearing capacity and the distance between the structural elements of the present facade and the panels,
  - the metal profiles, which are used when the structural elements of the present facade cannot support the load of the new envelope. They are fixed onto the metal fasteners and the panels are hung on them. There is only one type of metal profile but its length has to fit the facade height.

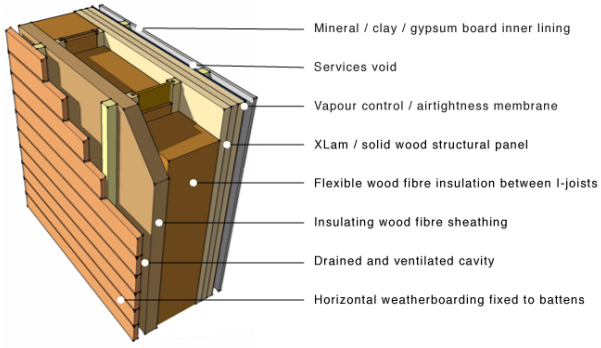


Figure 1: CRIBA Panels

The assembly process consists in a set of tasks to be carried out in order to assemble the new frame and envelope all around the building. It comprises some tasks that have always to be carried out, such as positioning and fixing metal fasteners, and some that are optional, such as fixing the metal profiles onto the metal fasteners.

At least, the configurator will give an idea of the renovation global cost which includes the costs of raw materials, transportation, labour and lifting devices.

On the first hand, in the configuration community, many authors (among them [Sabin and Weigel, 1998], [Soininen *et al.*, 1998]) have shown that product configuration could be efficiently modelled and aided when considered as a Constraints Satisfaction Problem (CSP) [Montanari, 1974]. On the other hand, in the civil engineering community and in the constraints community, many authors ([Honda and Mizoguchi, 1995], [Junker, 2006], [Medjdoub and Yannou, 2001], [Zawidzki *et al.*, 2011] or [Regateiro *et al.*, 2012]) have shown that spatial layout can be solved with CSP. Consequently, we address the building renovation configuration with constraints and filtering algorithms.

3 Generic BOM Model

In this paper, we focus on the interactive definition of the renovation bill of material. In this section, we highlight the main variables of the configurable components of the renovation BOM. We focus in this paper on the panels and the angles. At the end of the configuration, all the configurable components variables have a single value, given either by the user or deduced by the configurator.

3.1 Panels

The panels are rigid 2D rectilinear rectangles. That means that their sides are parallel to the facade reference axis. Let us consider one facade. All the panels covering it belong to a unique vertical plane. They are adjacent (they are at least one side in common) and are not overlapping themselves. By the way, they have all the same orientation. They cover completely the facade and make a partition of it.

The main variables of a panel refer to:

- its width $[min_w, max_w]$,
- its length $[min_l, max_l]$,
- its coordinates (abscissa and ordinate),
- its insulation thickness $[min_i, max_i]$,
- the insulation type (mineral wool or cellulose),
- its weight, which depends on its dimensions, the insulation type, and the components that are included in itself.

If the panel includes other components (windows, doors or solar modules), we need to know exactly for each of them:

- its width $[min_w, max_w]$,
- its length $[min_l, max_l]$,
- its relative position on the panel (x and y),
- its type and reference code.

A minimal distance is required between the sides of the panel and the position of other components: the distance cannot be lower than $90\ mm$ in order to preserve the panel stiffness.

3.2 Angles

The prefabricated angles are rigid 3D L-polyomino tetracubes which are placed at the building corners. The corners are at the intersection of two consecutive and perpendicular facade planes. We assume then that the angles are right. Otherwise, a specific design task must be carried out in order to design the relevant angles.

Let us considering a corner. All the angles covering it belong to a unique vertical row. They are adjacent (they are at least one side in common) and are not overlapping themselves. By the way, they have all the same orientation. They covered completely the corner and make a partition of it.

The main variables of an angle refer to:

- its width $[min_w, max_w]$,
- its right length $[min_{rl}, max_{rl}]$,
- its left length $[min_{ll}, max_{ll}]$,
- its coordinates (abscissa and ordinate),
- its insulation thickness $[min_i, max_i]$,
- the insulation type (mineral wool or cellulose),
- its weight, which depends on its dimensions and the insulation type.

For the first version of the BOM, the prefabricated angle cannot include other components. The angles dimensions directly depend on the sizes of their adjacent panels, with a minimal length (right and left) equal to $90\ mm$ in order to preserve the angle stiffness.

4 Building Renovation Configuration Process

The building renovation configuration is a top-down and multi-step process, which progressively converges from the working site to the configurable components. The user has to input some information and data in order to configure the renovation. After each user input, the configurator removes inconsistent values and guides progressively the user to a consistent solution. The user has to follow this process and gives information on:

- the whole working site. The working site description has an impact on the panels dimensions. Let us focus on the working site accessibility and its local atmosphere. Concerning its accessibility, if you can access the working site with special convoys, the panels can be as wide and long as needed. Otherwise, the dimensions of the panels are constrained by the size of the trucks which can access to the working site. Concerning the local atmospheric, if the working site area is very windy, wind speed peaks $\geq 80\ km.h^{-1}$, the panels have to be smaller in order to be fixed onto the faades, and the renovation lasts longer because of nonworking periods.
- the block of apartment buildings. The block description has directly an impact on the hoisting equipment and indirectly on the panels dimensions. If the block cannot be accessible with a tower crane, the panels have to be smaller in order to be conveying to the faades with an other suitable hoisting equipment, such as a telescopic boom lift.
- the apartment building. The apartment building description has an impact on the panels dimensions. Let us consider only the apartment building height. If the apartment building height is lower than twelve meters (four stories), the height of the panels can be the same as the building one so that the panels are fixed vertically on the faades.
- the faades. Let us focus on a facade.
  1. First of all, the user has to describe precisely the structure and the geometry of the facade. Considering the structure of the facade, (s)he needs to input where the metal fasteners can be fixed on the facade. A structural study has to be done in order to characterize the load bearing capacity of every area of the facade. Considering the geometry, the positions of windows, doors and balconies have to be known precisely. Only a topographic survey can provide these information.
  2. Regarding these areas and their characteristics, the decision of fixing the panels directly on the facade or on the metal profiles can be made. This decision has an impact on the BOM (metal fastener type and optional metal profiles) and on the assembly process (tasks devoted to metal profiles, such as delivery, assembly and adjustment).
  3. Having information about the working site, the block, the apartment building and the facade and the impact on the panels dimensions, the drawing of the facade layout can start. The user has now to

indicate what the aesthetic effect she/he wants for the facade. For instance, she/he can want continuous vertical joints, which means that the panels are fixed vertically or she/he can want a checkerboard effect with a lot of similar panels.

4. Knowing the facade layout, each panel has to be configured. If the panel is solid, one can decide to add solar modules or to add an exit door. If the panel has to include windows, doors and/or balconies, the suitable reference code has to be selected for each of them.

- the angles. The renovation configuration finishes by the configuration of the angles. At this step, only the height of angles has to be determined.

At any time in every step of the configuration process, the user can change her/his inputs and see their impact on the configuration solutions.

5 Building Renovation Configuration and Constraints

Interactive renovation configuration is provided by constraint propagation that prunes bad solutions and progressively guides the way to good ones. In apartment buildings renovation, the range of knowledge to exploit and to model leads us to integrate into a single configurator different constraints types as well as their filtering methods. In this section, we outline the kind of variables and constraints that are necessary to formalize apartment building renovation model.

5.1 Classical CSP Approaches

In building renovation configuration, we have to formalize different kinds of knowledge relevant to:

- civil engineering regulations that must be followed absolutely to the letter. For instance, fire barriers have to be installed between two consecutive stories in order to stop the spread of fire,
- civil engineering know-how that is the core knowledge of the companies involved in the *CRIBA* project,
- working site assembly process that allows us to define the suitable way of assembling the new frame and envelope all around the building.

For instance, we have seen in Section 4, that the working site local atmosphere has an impact on the panels dimensions: if the working site area is very windy, wind speed peaks ≥ 80 $km.h^{-1}$ several times a year, the panels have to be smaller in order to be fixed onto the faades without stopping the renovation with nonworking periods.

As the range of knowledge to model is wide, we need to use different CSP approaches and their filtering algorithms, such as discrete CSP ([Montanari, 1974], [Mackworth, 1977], [Bessire and Cordier, 1993], [Faltings, 1994]), continuous CSP ([Lhomme, 1993] or [Benhamou *et al.*, 1994]) and mixed CSP ([Gelle, 1998]) depending on the type of the variables (discrete, continuous, symbolic or numeric) and the type of constraints (compatibility constraints or mathematical formulae).

5.2 Groups and Multi-instances of Constraints

In the renovation configuration, we have to cope with several instances of the same configurable components. For instance, in order to cover a facade with its new envelope, we need to configure x times a panel (such as described in Subsection 3.1). We do not know in advance how many panels will be necessary, as it depends on a lot of data (working site description, block description, etc.). Therefore, we need to group variables and constraints into sets or classes, which can be instantiated as much as needed.

5.3 Dynamic Constraints

We have seen in the building renovation configuration process, Section 4, that we can decide to fix the panels on a metal envelope, or to create new openings on a facade. These decisions imply firstly the consideration of new components in the BOM and secondly, the insert of their assembly tasks in the assembly process. Therefore, we need to take into account the activation of configurable components as defined by [Mittal and Falkenhainer, 1990], [Sabin and Freuder, 1996] and [Faltings *et al.*, 1992].

5.4 Geometric Constraints

In order to prefabricate the panels, we need to determine precisely the dimensions and the position of each component on the panels. The accuracy of the topographic measures and the precision of the components dimensions and position are the crucial factors for the industrialization of the building renovation and the goals of the *CRIBA* project. Therefore, in order to do so, we need to integrate to the configurator geometric constraints (for a complete survey, see, [Dohmen, 1995] or [Fudos and Hoffmann, 1997], and for more recent work see [Jermann *et al.*, 2000], [Zawidzki *et al.*, 2011] or [Regateiro *et al.*, 2012]).

5.5 Global Constraints

As we cannot know in advance how many panels are needed to cover a facade, we have to cope with constraints that depend on the number of instances of the same class. For instance, if the height of the facade is covered with more than one panel, the sum of all the panels heights has to be equal to the facade one. Therefore, we need to integrate and filter different kinds of global constraints [van Hoeve and Katriel, 2006].

6 Conclusion

The aim of this paper has been to present a prospective study on the development of the interactive configuration system for the renovation of apartment buildings.

Firstly, we have presented the apartment buildings renovation problem and what the main objectives of the *CRIBA* configurator are: the interactive definition of the renovation thanks to the associated bill of material and the building site assembly process as well as a first cost estimation. Then we have focused on two configurable components that are the panels and the angles and highlighted their main characteristics. In the fourth section, we have outlined the top-down and multi-step building renovation configuration process. In

the final section we have put forward some ideas on the different kinds of CSP approaches we have to integrate in the configurator in order to support and guide the configuration of buildings renovation.

The apartment buildings renovation configuration is a challenge however you look at it. First of all, we want to industrialize a process that is nowadays traditionally made by craftsmen. This point is quite a revolution for the civil engineering field where only few industrial engineering methods are applied, and in particular in SMEs. Secondly, in order to be able to use a configurator, we need to extract, validate and formalize relevant knowledge. In our application field, the nature and the origin of knowledge are quite various. We have therefore to use different types of variables and constraints. The filtering engine has therefore to integrate different kinds of propagation methods. Thirdly, we need to cope with different variables priorities. For instance, all the variables which describe the whole working site have a strong impact on the dimensions of the panels and cannot be changed: we cannot decide to use a special convoy if the working site is not accessible with such a convoy. If an inconsistent solution is found, we will propose to the user to change her/his choices firstly on the panels and then to progressively zoom out to the whole working site.

As we are still in the very earliest stage of the *CRIBA* project and as apartment building renovation configuration is quite a complex process, we need to model in details the BOM components and their constraints. When this is done, we have to select, analyse, adapt and integrate constraints approaches and filtering algorithms in our propagation engine *CoFiADe*. *CoFiADe* has already been used for supporting heat treatments configuration [Aldanondo *et al.*, 2006], simultaneously product and planning configurations [Vareilles *et al.*, 2008] and helicopters maintenance configuration [Vareilles *et al.*, 2009]. The development of the graphical user interface in order to allow the user to see the result of her/his configuration is also a challenge. It is not the core of the configuration problem but it is clearly a need for the companies involved in the project.

Acknowledgements

The authors wish to acknowledge the *TBC Générateur d'Innovation* company, the *Millet* and *SyBois* companies and all partners in the *CRIBA* project, for their involvement in the construction of the CSP model.

References

[Aldanondo *et al.*, 2006] M. Aldanondo, E. Vareilles, K. Hadj-Hamou, and Paul Gaborit. A constraint based approach for aiding heat treatment operation design and distortion evaluation. In *Artificial Intelligence Applications and Innovations*, volume 204 of *IFIP International Federation for Information Processing*, pages 254–261. Springer US, 2006.

[Benhamou *et al.*, 1994] F. Benhamou, D. Mc Allester, and P. Van Hentenryck. Clp(intervals) revisited. In *ILPS'94*, pages 1–21, 1994.

[Bessire and Cordier, 1993] C. Bessire and M.O. Cordier. Arc-consistency and arc-consistency again. In *AAAI*, pages 108–113, 1993.

[Center, 2012] The Energy Conservation Center. *Energy Conservation Handbook*. Japan, 2012.

[Council, 2013] U.S. Green Building Council. *New Construction Reference Guide*, 2013.

[Dohmen, 1995] Maurice Dohmen. A survey of constraint satisfaction techniques for geometric modeling. *Computers & Graphics*, 19(6):831–845, 1995.

[Falcon and Fontanili, 2010] M . Falcon and F . Fontanili. Process modelling of industrialized thermal renovation of apartment buildings. *In eWork and eBusiness in Architecture, Engineering and Construction, European Conference on Product and Process Modelling (ECPPM 2010)*, September 2010.

[Faltings *et al.*, 1992] B. Faltings, D. Sam-Haroud, and I. Smith. Dynamic constraints propagation with continuous variables. *European Conference on Artificial Intelligence*, pages 754–758, 1992.

[Faltings, 1994] B. Faltings. Arc consistency for continuous variables. In *Artificial Intelligence*, volume 65, pages 363–376, 1994.

[Felfernig, 2007] A. Felfernig. Standardized configuration knowledge representations as technological foundation for mass customization. In *IEEE Transactions on Engineering Management*, volume 54, pages 41–56, February 2007.

[Fudos and Hoffmann, 1997] I. Fudos and C. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.

[Gelle, 1998] E. Gelle. *On the generation of locally consistent solution spaces in mixed dynamic constraint problems*. Thse de doctorat, École Polytechnique Fdrale de Lausanne, Suisse, 1998.

[Honda and Mizoguchi, 1995] K. Honda and F. Mizoguchi. Constraint-based approach for automatic spatial layout planning. In *Proceedings of the 11th Conference on Artificial Intelligence for Applications*, CAIA '95, Washington, DC, USA, 1995. IEEE Computer Society.

[Jermann *et al.*, 2000] C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A constraint programming approach for solving rigid geometric systems. In *Constraint Programming*, Singapore, 2000.

[Juan *et al.*, 2010] Y.K. Juan, P. Gao, and J. Wang. A hybrid decision support system for sustainable office building renovation and energy performance improvement. *Energy and Buildings*, 42(3):290–297, March 2010.

[Junker, 2006] U. Junker. *Handbook of Constraint Programming*, chapter Chapter 24. Configuration. Elsevier, 2006.

[Lhomme, 1993] O. Lhomme. Consistency techniques for numeric CSP. In *International Joint Conference on Artificial Intelligence*, pages 232–238, Chambry, France, Aot 1993.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, volume 8(1), pages 99–118, 1977.

[Medjdoub and Yannou, 2001] B. Medjdoub and B. Yannou. Dynamic space ordering at a topological level in space planning. In *Artificial Intelligence in engineering*, volume 15, pages 47–60, January 2001.

[Mittal and Falkenhainer, 1990] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *AAAI*, pages 25–32, Boston, US, 1990.

[Montanari, 1974] U. Montanari. Networks of constraints: fundamental properties and application to picture processing. In *Information sciences*, volume 7, pages 95–132, 1974.

[Perez-Lombard *et al.*, 2008] L. Perez-Lombard, J. Ortiz, and C. Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3):394 – 398, 2008.

[Poel *et al.*, 2007] B. Poel, G. van Cruchten, and C.A. Balaras. Energy performance assessment of existing dwellings. *Energy and Buildings*, 39(4):393–403, April 2007.

[Regateiro *et al.*, 2012] F. Regateiro, J. Bento, and J. Dias. Floor plan design using block algebra and constraint satisfaction. *Advanced Engineering Informatics*, 26(2):361–382, April 2012.

[Sabin and Freuder, 1996] D. Sabin and E.C. Freuder. Configuration as composite constraint satisfaction. In *Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[Sabin and Weigel, 1998] D. Sabin and R. Weigel. Product configuration frameworks a survey. In *IEEE Intelligent Systems*, volume 13, pages 42–49, 1998.

[Soininen *et al.*, 1998] T. Soininen, T. Tiihonen, T. Mnnist, and R. Sulonen. Towards a general ontology of configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):357–372, 1998.

[van Hoeve and Katriel, 2006] Willem-Jan van Hoeve and Irit Katriel. *Handbook of Constraint Programming*, chapter Chapter 6. Global Constraints. Elsevier, 2006.

[Vareilles *et al.*, 2008] E. Vareilles, M. Aldanondo, M. Djefel, and P. Gaborit. Coupling interactively product and project configuration: a proposal unsing constraints programming. In *International Mass Customization and International Conference on Economic, Technical and Organisationel Aspects of Product Configuration Systems*, June 2008.

[Vareilles *et al.*, 2009] E. Vareilles, C. Beler, E. Villeneuve, M. Aldanondo, and L. Geneste. Interactive configuration and time estimation of civil helicopter maintenance. In *Workshop on Configuration in the European International Joint Conferences on Artificial Intelligence*, Los Angeles, California, USA, July 2009.

[Zawidzki *et al.*, 2011] M. Zawidzki, K. Tateyama, and I. Nishikawa. The constraints satisfaction problem approach in the design of an architectural functional layout. *Engineering Optimization*, 43(9):943–966, 2011.

Configuration Dynamics Verification Using UPPAAL

David Fabian
Dept. of Mathematics
Faculty of Nuclear Sciences and
Physical Engineering
Czech Technical University in Prague
(fabiadav@fjfi.cvut.cz).

Radek Mařík
Dept. of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague.

Abstract

Modern software applications can have very complicated internal dynamics. Most of the software tools are written in an imperative programming language which can quickly become impractical for describing complex dynamics. Also, it is very hard to verify that the code actually covers fully all aspects of the tool's dynamics. Propagation rules are suitable as a means for specification and verification of such dynamic systems. We have selected a software tool from the domain of configuration for our study. Configuration wizards and tools are examples of software applications where even a small change made by the user can lead to a very complex outcome. In this paper, a configuration hierarchical model and a syntax of propagation rules are introduced. These constructs can be used to describe declaratively the dynamics that is typical for software configuration tools. The hierarchical model is then used for describing the internal dynamics of the configuration tool Freeconf. This specific model instance is then implemented in UPPAAL and verified by the UPPAAL model-checker.

1 Introduction

Software applications become more and more complicated, nowadays. The complexity of the internal dynamics of a modern software application can be hard to maintain. Software configuration is one of the areas where the dynamics can become very complicated.

Software configuration can be divided into two distinct groups. In literature, configuration is usually understood as finding such a combination of software/hardware modules that the resulting product satisfies some prescribed requirements [Vlaeminck *et al.*, 2009]. On the other hand, from the point-of-view of the end-user, configuration process means changing some options (configuration keys) of a finished product, so that it will adapt to the user's needs (changing the background of the desktop, choosing the size of the subtitles in the media-player, setting up permissions for the web server) [Liaskos *et al.*, 2005; Fabian, 2012]. Nowadays, there exist software tools designed to aid the user with these application adjustments. They often offer a GUI (Graphical User Interface) in a form of one or more configuration windows and are usually hard-wired to the application itself. There also exist general-purpose configuration tools such as KConfigXT [TechBase, 2012], and Freeconf [Fabian, 2012].

Since there can be many possible configuration keys in a configuration, it is natural to organize the keys into hierarchical categories. Each key also has an inner state formed by some properties that describe it. When the user interacts with a configuration tool, some configuration keys change their state in reaction to the user's input. Every change can be propagated further across the hierarchy and induce more changes in other keys depending on the semantics of the properties. In a configuration tool with many internal key properties, the amount of property interactions can lead to a complex dynamical behavior which is difficult to implement in an imperative style programming language. However, it is straightforward to describe the dynamics in a declarative form as propagation rules. The elegance of this approach consists in condensing the description of the dynamics (which can be very complex) to a single list of rules. That list can be then (semi)automatically verified for its soundness and completeness.

This topic is related to problems from the domain of production rules and knowledge bases [Arman, 2013; Preece and Shinghal, 1994; Preece and Shinghal, 1992]. Some initial work has been done to address the problem of automatic detection of rules redundancy and inconsistency in [Lukichev, 2011]. In the paper, the author uses description logic [Nardi and Brachman, 2003] to describe, in an abstract and general way, some typical patterns which can lead to an inconsistent or non-minimal set of production rules. The long-term goal of our work, however, is to develop a usable software application which would (semi)automatically verify the minimality and consistency of hierarchical models that are used in software configuration tools.

In this paper, an attempt to model and verify configuration software dynamics is introduced. A general model of a configuration hierarchy is described together with declarative rules that are used to model the dynamics on top of the hierarchy. Further, it is shown on a specific instance of the model (which is used in Freeconf) how such a set of rules can look like. Finally, the soundness of this instance is studied and verified by using the model-checking utility UPPAAL.

The rest of this paper is divided as follows. In Section 2, the hierarchical model and propagation rules are presented. Section 3 describes the properties used in Freeconf and the specific set of rules that describe the dynamics of properties propagation in it. Section 4 introduces UPPAAL, a model-checking software tool. In Section 5, it is presented how the Freeconf propagation rules can be modeled and their soundness verified in UPPAAL. Finally, Section 6 summarizes the results and Section 7 concludes.

2 Hierarchical Model and Rules

In this section, a configuration hierarchical model and a syntax of propagation rules will be introduced.

2.1 Hierarchical Model

The hierarchical configuration model can be thought as a rooted acyclic graph where each node has an internal state. The state of

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

a node can be changed directly by the environment (i.e., the user) or indirectly as a result of propagation of some direct change. The internal state will be limited to only Boolean and bounded integer properties in this paper.

Definition The *internal state* of a node in the hierarchical configuration model is a tuple of sets (B, I, D), where $B_i \in B$ represents a Boolean property and $I_j \in I$ represents an integer property from a single bounded integer domain D. At least one of the sets B and I must be non-empty.

Definition A *node* of a configuration hierarchical model is a tuple (i, p, C, X), where i is a unique positive integer index, p is the node's parent index, C is a (possibly empty) set of indices of the node's successors in the graph, and X is the internal state of the node. There exists a special parent index $\emptyset$ for the top-level node of the hierarchy denoting the absence of a parent.

Because some properties in different nodes can have the same name, a term X_j^i will be used henceforth to denote a property X_j of the node with index i.

Definition A *configuration hierarchical model* M is a non-empty set of nodes. A top-level node, i.e. the one with the parent index $\emptyset$ must always be present.

2.2 Propagation Rules

In general, the user can initiate a propagation by changing any property of an arbitrary node in the hierarchy (even more properties at once). From that, based on the semantics of the propagation, the change can propagate within that node, further up to the node's parent, down to its successors, or does not have to propagate at all.

The dynamics can be formally described by propagation rules.

Definition A *propagation rule* has a form $\mathcal{A} \rightarrow \mathcal{B}$, where $\mathcal{A}$ is the head (condition) of the rule and $\mathcal{B}$ is the body (action). The head is always bound to a specific node and consists of a non-empty conjunction of the node's Boolean properties or their negations and terms $I_j \circledast v_j$, where I_J is the node's integer property and v_j is an integer constant. $\circledast$ is a substitute for comparison operators $<, >, \leq, \geq, ==, \neq$. The body is formed by a non-empty conjunction of assignments to, in general, Boolean and integer properties of the node itself, to properties of the node's parent and to properties of its children. If the head of a propagation rule is satisfied (i.e. all Boolean properties are true and all comparison operations hold) the rule fires and the body is executed leading to a change of values of other properties.

The terms "head" and "body" are used in this particular order to match Constraint Handling Rules (CHR) terminology since there are plans to use CHR to develop propagation rules solver (see Section 7).

A syntactical shortcut will be used to express $I_j = I_j + 1$ and $I_j = I_j - 1$, i.e., incrementing and decrementing an integer property by one, as I_j++ and I_j--, respectively. According to the definition, a general propagation rule that operates only within a single node with index a will have the following form

$$\left(\bigwedge_i B_i^a \wedge \bigwedge_m (I_m^a \circledast v_m)\right) \rightarrow \bigwedge_j \left(B_j^a = b_j\right) \wedge \bigwedge_k \left(I_k^a = c_k\right) .$$

A propagation rule that represents a communication between a node and its parent will be

$$\left(\bigwedge_i B_i^a \wedge \bigwedge_m (I_m^a \circledast v_m)\right) \rightarrow \bigwedge_j \left(B_j^p = b_j\right) \wedge \bigwedge_k \left(I_k^p = c_k\right) .$$

Finally, a propagation rule that describes a communication between a node and some of its children can be described as

$$\left(\bigwedge_i B_i^a \wedge \bigwedge_m (I_m^a \circledast v_m)\right) \rightarrow \bigwedge_{j,l} \left(B_j^{C_l} = b_j\right) \wedge \bigwedge_{k,l} \left(I_k^{C_l} = c_k\right) .$$

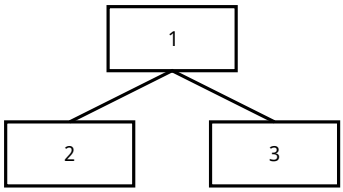


Figure 1: An example of configuration hierarchical model with three nodes.

To better illustrate how different types of propagation can look like, let us consider a configuration hierarchical model M with three nodes as given in Figure 1. Each node will have an identical structure of the internal state $X = (bool_1, bool_2, int_1, \{0, 1, 2\})$. The model M will be a set of three tuples

$$\begin{aligned} M = \{\ & \left(1, \emptyset, \{2, 3\}, \left(bool_1^1, bool_2^1, int_1^1, \{0, 1, 2\}\right)\right), \\ & \left(2, 1, \emptyset, \left(bool_1^2, bool_2^2, int_1^2, \{0, 1, 2\}\right)\right), \\ & \left(3, 1, \emptyset, \left(bool_1^3, bool_2^3, int_1^3, \{0, 1, 2\}\right)\right)\ \} . \end{aligned}$$

Let us further assume that the semantics of the properties declares that:

- whenever $bool_1$ is $false$ for node two, $bool_2$ must also be $false$ for that particular node
- whenever $bool_2$ is $true$ and int_1 is greater than one in node three, the value of the parent's int_1 must be two
- whenever int_1 is zero for node one, $bool_2$ for node two must be $true$ and int_1 for node three must be one

The respective propagation rules are given below.

$$\begin{aligned} & \neg bool_1^2 \rightarrow bool_2^2 = false \\ & bool_2^3 \wedge int_1^3 > 1 \rightarrow int_1^1 = 2 \\ & int_1^1 == 0 \rightarrow bool_2^2 = true \wedge int_1^3 = 1 \end{aligned}$$

Of course, the body of a propagation rule can affect not only the node itself, the parent, and the successors separately, but also any combination of the respective internal states. In general case, poorly designed rules can form a loop and thus lead to a non-terminating computation.

3 Freeconf properties

In this section, Freeconf is briefly introduced and its internal dynamics modeled as a configuration hierarchical model is presented.

3.1 Freeconf Tool

Freeconf is a general-purpose cross-platform configuration utility developed at Czech Technical University [Fabian, 2011; Fabian, 2012]. The tool is supposed to create an intermediate layer between the user and an application without any configuration GUI. An example of such applications can be various application servers, web servers, some movie players, and basically any program that stores its configuration in configuration text files. When requested by the user, Freeconf automatically generates a configuration dialog

with application specific configuration options (configuration keys), the user then can change the configuration according to her liking, and during saving Freeconf transforms the output into the respective native configuration files from where the application can read the changes. The details about this process can be found in [Fabian, 2012].

3.2 Key and Section Properties

property	meaning
static mandatory (*sman*)	If it is false, the key is never shown to the user unless *show all* property is set. If it is true the visibility is controlled by the dynamic equivalent of this property.
static active (*sact*)	If it is false, the key is as being commented out from the list of possible keys. No further action is applicable to the key and the key is not visible to the user.
dynamic mandatory (*dman*)	This property can only be set as a result of propagation of the user's action. If it is true, the key is mandatory and must be shown. This property has no meaning when the static mandatory property is set to false.
dynamic active (*dact*)	This property can only be set as a result of propagation of the user's action. If it is false, the key does not have sense in the current settings. This property has no meaning when the static active property is set to false.
value set (*valset*)	This property is true iff the key has a value assigned. In some configuration, there can be an initial state where the key does not have any value.
default value set (*defvalset*)	This property is true iff the key has a default value assigned. There can be some keys in the configuration which do not have a default values assigned.
undefined (*undef*)	This is an aggregation property which is true iff the value and default values are both unset.
inconsistent	The key is inconsistent with the configuration iff it is undefined and is dynamically mandatory and dynamically active.
show all (*showall*)	This is a special property which overrides whether the keys are shown to the user or not. Every key will always have the same value of this property because the user will change it at once for all the keys (broadcast change). If this property is true all dynamically active keys are shown to the user, even those that are not mandatory.

Table 1: Freeconf key properties.

Freeconf can handle hundreds or even thousands of configuration options. To avoid overfilled and confusing configuration dialogs, it is necessary to divide the options into specific categories. It is done by assigning a set of Boolean properties to every option such that truthfulness of a specific set of properties means the option belongs to the respective category. At the moment, Freeconf uses, apart from the basic set of properties that are static and do not participate in property propagation, a set of properties for every configuration key. *Static* properties are meant to be fixed throughout the run of Freeconf and cannot be changed by the user. *Dynamic* properties, on the other hand, can have their values changed in reaction to the user's action quite often. *Mandatory* property reflects the fact that the key is vital for the configuration and must be shown to the user at any case. *Activity* of the key determines its current state of presence or absence in the configuration. The key properties are given in Table 1.

Semantically related configuration keys are often grouped together to so called *configuration sections*. These are basically containers which can hold both other configuration sections or configuration keys. The sections have themselves some properties that help them to keep track of the state of their direct successors and react, for example, to the situation where all successors of a given section should be hidden. In that case, the section should hide itself too. The current set of section properties is given in Table 2.

Freeconf has a semantics which describes the evolution of properties values in reaction to the users actions. It has been formulated as a set of propagation rules in [Fabian *et al.*, 2012].

In the context of Section 2, it is easy to encode Freeconf properties propagation into a configuration hierarchical model. There will be two types of nodes in the model, one for configuration keys and one for configuration sections. Following Definition 2.1, the internal state of a configuration key will be a tuple with nine Boolean properties and no integer properties:

$$\begin{aligned} X = (\ &\{defvalset, valset, sman,\\ &dman, sact, dact,\\ &undef, inconsistent, showall\},\\ &\emptyset,\\ &\emptyset\ )\ . \end{aligned}$$

The internal state of a configuration section will be a tuple of two Boolean properties and four integer properties:

$$\begin{aligned} Y = (\ &\{empty, inconsistent, showall\},\\ &\{mancounter, actcounter,\\ &\ inccounter, sectionshowncounter\},\\ &[0 \ldots N]\ )\ , \end{aligned}$$

where N is the number of successors of the section. The configuration hierarchy will be formed with configuration keys as leaf nodes and configuration sections as non-leaf nodes.

3.3 Propagation Rules

The list of all propagation rules that describe the dynamics in Freeconf is given in [Fabian *et al.*, 2012] where the rules are represented using a rule-based constraint programming syntax [Brand, 2004]. A transformation to the propagation rules syntax is straightforward. For example, whenever *dynamic mandatory* property of a node changes its value, the parent must be informed and its *mandatory counter* must be adjusted accordingly. Propagation rules describing this change are shown below.

$$\begin{aligned} dman_i &\rightarrow mancounter_i^p\text{++}\\ \neg dman_i &\rightarrow mancounter_i^p\text{--} \end{aligned}$$

In Freeconf, rules rewrite only nodes that are higher up in the hierarchy, i.e., node to section and section to section communication. This flow of information suffices for the needs of Freeconf and prevents non-termination.

property	meaning
mandatory counter (*mancounter*)	This counter reflects the number of key successors that are dynamic mandatory.
active counter (*actcounter*)	This counter reflects the number of key successors that are dynamic active.
inconsistent counter (*inccounter*)	This counter reflects the number of successors (even sections) that are inconsistent.
section shown counter (*sectionshowncounter*)	This counter holds the number of successor sections that are not empty.
empty	This property is true iff there are no mandatory and active key successors and no non-empty successor sections.
inconsistent	This property is true if there is at least one inconsistent successor. In another words, the inconsistent counter is not zero.
show all (*showall*)	The same property as in the case of the keys. The user will change the value for every section and every key at once overriding the emptiness of the section.

Table 2: Freeconf section properties.

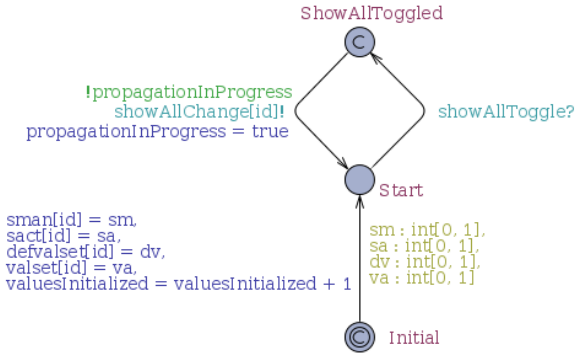


Figure 2: A state machine automaton template designed in UPPAAL.

modality	meaning
E <> φ	There exists a run in which φ eventually holds.
A <> φ	In every run, φ eventually holds.
E[]φ	There exists a run in which φ always holds.
A[]φ	In every run, φ always holds.

Table 3: Modalities used in UPPAAL.

4 UPPAAL

UPPAAL is a model-checking verification software of real-time dynamic systems developed by Upsalla University and Aalborg University [Behrmann *et al.*, 2004; David *et al.*, 2009]. A modeled system is represented as a network of timed automata and one can verify the soundness of the model by querying the built-in model checker.

4.1 Modeling in UPPAAL

UPPAAL offers a GUI written in Java to design each automaton of the network graphically. One can also program parts of the automata using a C-like syntax language. A very useful feature is templating which simplifies designing of very similar automata by using constant template parameters. UPPAAL will automatically unfold a template by creating an automaton according to the template for every possible value of the template parameter.

UPPAAL further supports adding guards to transitions, time invariants to nodes, choosing non-deterministically a value of a variable during a transition, and a synchronization of two or more concurrent transitions via signals. An example of a modeled automaton template can be seen in Fig.2.

UPPAAL offers declaration of constants, typed variables and single-dimensional and multi-dimensional arrays. A variable can be either Boolean, or a bounded integer (in fact, Boolean is a special case of `int[0,1]`). The scope of a variable is either local to the automaton, or is global, so all parts of the model can access the variable. In Fig. 2, the variable `sm` is Boolean and its value is chosen randomly upon transition from the `Initial` state to the `Start` state. During that same transition, the value of `sm` is assigned to the cell of array `sman` with index `id`. The variable `id` is in fact a template parameter and its value is different yet constant in every instance of this template.

4.2 Query Language

The main purpose of a model-checker is to verify the model w.r.t. a requirement specification [Behrmann *et al.*, 2004]. The requirements must be formulated in a well-defined language. UPPAAL uses a simplified version of timed computation tree logic (TCTL) [Alur *et al.*, 1993].

$$\text{E} <> forall(i : id\_s)manCounter[i] < 0 \quad (1)$$

An example of a query is given in Equation 1. The query usually starts with a path quantifier and a modality determining the validity of a formula along a specific run of the system. All possible modalities are presented in the following table.

Apart from the modalities stated above, UPPAAL also supports the *until* modality φ --> ψ which can be read as "In every run, if φ holds then ψ eventually holds". The same formula can be obtained by using only the modalities from Table 3 A[](φ => (A <> ψ)). UPPAAL however does not support multiple modalities in a single query, so --> is the only possibility.

The model-checker in UPPAAL is written in C and it is possible to choose different sub-algorithms it uses during the verification via the program's menu. If the verification of a query fails, UPPAAL can be set to produce a counter-example in the form of a system trace. The trace demonstrates how to get from the initial state to a state where the query does not hold.

5 Modeling of Rules Using UPPAAL

Since the hierarchical configuration model can become quite large and there can exist a lot of propagation rules, it would be profitable to have a means of verification that the rules express the intended semantics correctly, are consistent, and not redundant. The hierarchical model can be thought of as a Kripke structure where each instance of the model forms a possible world and propagation rules describe possible transitions. Since UPPAAL in its core uses Kripke structures, it is natural to model the configuration hierarchical model in UPPAAL.

As the first attempt, the Freeconf model was verified in UPPAAL. The entire Freeconf specific configuration hierarchical model was encoded into UPPAAL and then the model-checker was used to perform the verification. In the following sub-sections, each of the steps will be briefly described. The entire model and all

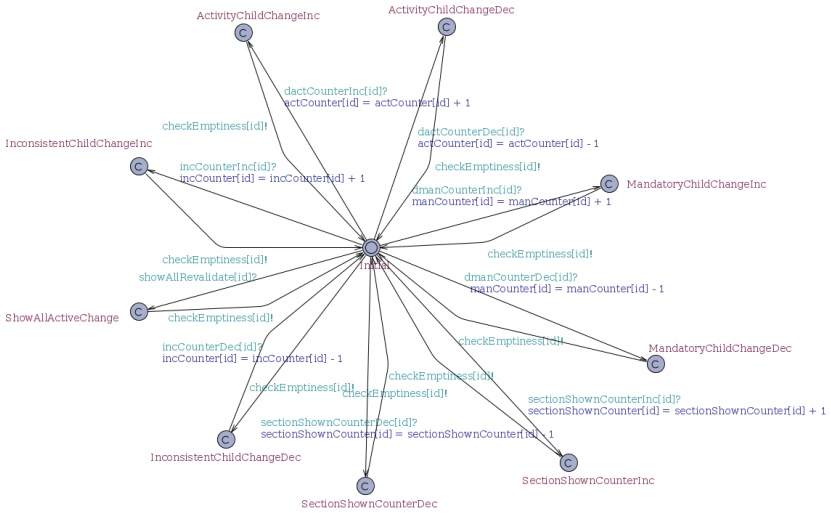


Figure 3: Freeconf section node modeled in UPPAAL.

queries can be found at `http://kmlinux.fjfi.cvut.cz/~fabiadav/phd/uppaal/freeconf_model.zip`.

5.1 Modeling Phase

There are multiple problems to be solved while modeling the hierarchy. Firstly, Freeconf key and section properties forming the internal states of Freeconf nodes have to be inserted. As shown in Listing 1, they are declared as global Boolean and integer arrays in a straightforward way, where each node of the hierarchical model is assigned a non-negative index and accessing the internal state of the node is simply done by reading elements from all the arrays with the same index. Even though in Freeconf, integer counters generally do not have the same domains, it is acceptable to approximate the general situation by setting the upper limit of each counter to the number of nodes in the hierarchy (to the number of sections in the hierarchy for *sectionShownCounter*). Property *show all* is implemented as a single shared Boolean variable for the user template to be able to change the property at once.

```
const int N = 3; // number of keys
const int M = 2; // number of sections

bool defvalset[N], valset[N], sman[N],
     dman[N], sact[N], dact[N], undef[N],
     inconsistent[N];

int[0, N] incCounter[M], manCounter[M],
          actCounter[M];
int[0, M] sectionShownCounter[M];
bool empty[M];
bool showAll;
```

Listing 1: Declaration of Freeconf properties

The hierarchical nodes are encoded as state machine automaton templates parametrized by node indices. For each key, there exists an automaton `Node` which was presented earlier in Figure 2. The automaton has two main tasks to perform — setting the initial state of all its properties to random values (so that the model-checker can later test all possible combinations of properties values) and updating the visibility in the case when the user has modified the show all property. For each section, there is an automaton `Section` given in Figure 3.

The section template merely listens to signals from its successors and updates its respective counters and the emptiness status.

The most complicated part deals with the design of the hierarchical model. Since channel synchronization and global variables are the only possibilities to exchange information between the automata in UPPAAL, properties propagation is implemented by using these two. Two separate tree connections have to be considered because they behave differently. For section-key connections (that is to model the propagation between the keys and their parent sections), a special automaton template `NodeSectionDispatcher` is created. For section-section connections (i.e., for modeling the propagation between the sections and their parent sections), another automaton template `SectionDispatcher` is created. `NodeSectionDispatcher` template is parametrized by an index which spawns from zero to the number of section-key pairs. `SectionDispatcher` template is similarly parametrized by an index going from zero to the number of section-section pairs (including the top-level pair whose one end is connected the top- level section and the other to `TopLevelTerminator` automaton). Two global two-dimensional arrays are declared which, in fact, model the hierarchical tree as a parent-child relation for node indices.

```
const int NODECON = 3;
const int SECTIONCON = 2;
const int[0, N] disIdx[NODECON][2] =
                {{0, 0}, {1, 0}, {2, 1}};
const int[0, M] disSecIdx[SECTIONCON][2] =
                {{0, 1}, {1, 2}};
```

Listing 2: Hierarchical tree modeled as two two-dimensional arrays

Array `disIdx` has key indices in its first dimension and their parent section indices in its second dimension. Thus, in the example above key zero has section zero as its parent, key one has section zero, etc. Array `disSecIdx` is the equivalent of `disIdx` array but this time it describes section relationship. Note that in the initialization of `disSecIdx`, constant two is used even though in this case, only two sections are present in the example model, and hence

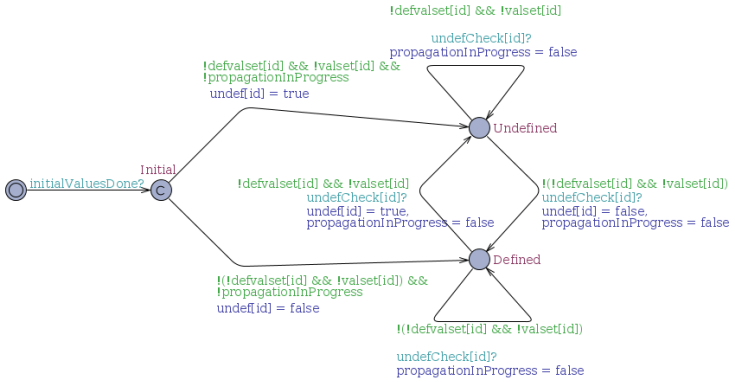


Figure 4: Propagation rules modifying the property *undefined* modeled as a state machine.

the index should be invalid. The first index that is not a valid section index is, however, used to denote the top-level connection and is thus valid in this context.

Constants `NODECON` and `SECTIONCON` represent the number of section-key connections and section-section connections, respectively. These arrays are used by the dispatcher automata to automatically dispatch propagation signals across the tree. This settings is flexible in the way that changing the shape of the hierarchy is simply a matter of adjusting four constants and two arrays.

While the tree structure can be abstracted and generalized to allow easy modifications, the actual rules have to be hard-wired into the automata templates because the semantics behind those rules must be expressed in a visual form. For example, the automaton presented in Figure 4 models the behavior of propagation rules that modify *undefined* property.

Enforcing causality of the propagation turns out to be another complication. For example, the initialization phase, where the keys are assigned some initial values and the first step of the propagation fires, must come before the user is activated. Enforcing causality does not require to use clocks that are the integral part of UPPAAL, only auxiliary variables and synchronization channels suffice. On the other hand, since automata execution is by default parallel in UPPAAL, the auxiliary code that controls causality renders the model more complicated and less clear.

Finally, the user is modeled as a single state machine automaton. The automaton non-deterministically chooses a key in the hierarchy and one of its properties that is changeable at the current state. In order to avoid race-conditions, every user's action must lock the hierarchy until the propagation has been finished. A global variable `propagationInProgress` is used for this purpose.

5.2 Verification Phase

Of course, since Freeconf model can become arbitrarily large by adding more sections and keys to the model, only a small amount of actual instances of the general Freeconf model is verified by the exemplar UPPAAL model. The instances are given in Figure 5. In reality, Freeconf can easily have four or five levels of sections in the hierarchy since there exist auxiliary non-visible sections and window tabs that act as sections in the model. On the other hand, Freeconf model is somehow homogeneous which means that if node to section and section to section properties propagation is valid than Freeconf model with arbitrarily large tree should be also valid.

Model-checking queries are divided into several groups. The first group serves a purpose of testing the UPPAAL model itself, because the encoding of the problem is not very straightforward and often

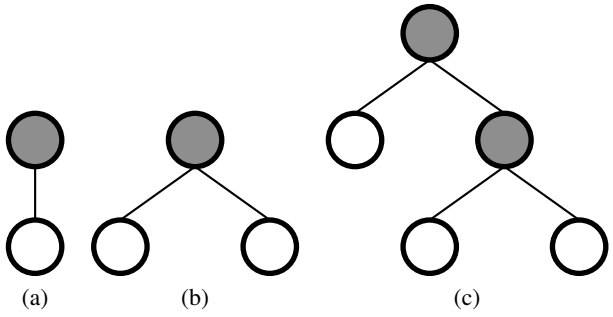


Figure 5: Freeconf hierarchical models validated in UPPAAL. Sections are depicted as gray, keys are white.

leads to cycles and non-termination. The basic liveness checking query $E <> deadlock$ appears to be very useful.

When the model is ready, it is necessary to further test whether updates to the integer properties do not set any value out of the domain. Queries similar to the one in the following listing are used for this test for every counter.

$$E <> forall(i : id_s)\ actCounter[i] > N$$
$$E <> forall(i : id_s)\ actCounter[i] < 0$$

Of course, just in this situation, the domains of the respective counters are changed to $[-1, N + 1]$ or $[-1, M + 1]$. Other basic type checking was also done.

In [Fabian *et al.*, 2012], one of the open problems was to determine if the following two sets of propagation rules that are given in Equation 2 and 3 are complementary and if one can replace the rule heads in the second set by just the negation of the heads from the first set. These two rules affect the state of the section *emptiness* property. They have got non-symmetric heads as a result of iterative development of Freeconf.

$$\begin{aligned}
&empty_i^i \wedge \Big(sectionshowncounter_i^i > 0\Big) \rightarrow \varphi \\
&showall_i^i \wedge \Big(activeshown_i^i > 0\Big) \rightarrow \varphi \\
&\neg showall_i^i \wedge \Big(activeshown_i^i > 0\Big) \wedge \\
&\wedge \Big(mandatoryshowncounter_i^i > 0\Big) \rightarrow \varphi \\
&\varphi := \Big(empty_i^i = false \wedge (sectionshowncounter_i^p - -)\Big)
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\neg empty_i^i \wedge \Big(mandatoryshowncounter_i^i == 0\Big) \wedge \\
&\wedge \Big(sectionshowncounter_i^i == 0\Big) \rightarrow \varphi' \\
&\Big(activeshowncounter_i^i == 0\Big) \wedge \\
&\wedge \Big(sectionshowncounter_i^i == 0\Big) \rightarrow \varphi' \\
&\varphi' := \Big(empty_i^i = true \wedge (sectionshowncounter_i^p + +)\Big)
\end{aligned} \tag{3}$$

By using a query (and its derivation with negated outer conjuncts) which are given in Equation 4, it can be shown that the rule sets are not complementary and there exist situations where both rules are applicable. To solve this error in the model, one has to modify the rule head in the second set by adding $\neg showall_i$ as is shown in Equation 5. After this minor tweak, the rules behave correctly and the *emptiness* property is set correctly in all situations with the existing code in Freeconf.

$$\begin{aligned}
&E <> forall(i : id_s)(sectionShownCounter[i]|| \\
&||(showAll\&\&actCounter[i])|| \\
&||(!showAll\&\&actCounter[i]\&\&manCounter[i]))\&\& \\
&\&\&((!manCounter[i]\&\&!sectionShownCounter[i] \\
&\&\&!showAll)||(!sectionShownCounter[i]\&\& \\
&\&\&!actCounter[i]))
\end{aligned} \tag{4}$$

$$\begin{aligned}
&\neg empty_i^i \wedge \Big(mandatoryshowncounter_i^i == 0\Big) \wedge \\
&\wedge \Big(sectionshowncounter_i^i == 0\Big) \wedge \underline{\neg showall_i^i} \rightarrow \varphi' \\
&\Big(activeshowncounter_i^i == 0\Big) \wedge \\
&\wedge \Big(sectionshowncounter_i^i == 0\Big) \rightarrow \varphi'
\end{aligned} \tag{5}$$

One of the hardest part is to construct a query that would allow us to ask for the correctness of propagation of the node and section *inconsistency* property. The final query which is shown below in Equation 6 must use UPPAAL node names and the *until* operator to be able to express the property update dynamics. Since UPPAAL does not allow universal quantification with the *until* operator, it is not possible to create a general query for every node.

$$\begin{aligned}
&(empty[0]\&\&empty[1]\&\&dman[0]\&\&dact[0]\&\& \\
&Inconsistent(0).SetInconsistentTrue) \\
&\text{-->}(!empty[0]\&\&!empty[1]\&\& \\
&TopLevelTerminator.TerminationDone)
\end{aligned} \tag{6}$$

5.3 UPPAAL Model-checker Performance

Because UPPAAL model is parametrized so that it can be very easily modified to represent a different instance of Freeconf model, it is interesting to measure UPPAAL's model-checker performance with respect to the size of Freeconf model. A fixed query 7 which tests the correctness of the node *inconsistency* property modifications is used for the needs of this measurement. The query uses "for all" path quantifier A and "always" temporal modality $[\,]$, so UPPAAL would have to traverse a great amount of states to draw a conclusion.

$$\begin{aligned}
&A[\,]forall(i : id_k)(undef[i]\&\&dman[i]\&\&dact[i]\&\& \\
&\&\&!propagationInProgress\&\&!userAction\&\& \\
&\&\&initFinished)\ imply\ (inconsistent[i])
\end{aligned} \tag{7}$$

In Table 4, the time and consumed memory it took to finish the validation process of the query for the three Freeconf models which were introduced in Figure 5 is given together with the upper estimate on the number of states UPPAAL has to traverse. The test was performed on a desktop PC with Intel Core 2 Quad Q9550 CPU at 2.83 GHz, 4 GiB RAM, running 64 bit Linux 3.1.10 and a development snapshot of UPPAAL 4.1.14.

Model	Time (s)	Memory (KiB)	# of states
a	0.07	6889	16384
b	1.67	24572	21233664
c	189.1	2147932	17592186044416

Table 4: Performance statistics of UPPAAL model-checker with respect to the size of Freeconf model.

6 Results

The entire Freeconf model has been encoded in UPPAAL. Some of the parts of the hierarchical configuration model, like the tree structure, are easy to implement in UPPAAL, others, like propagation rules, are more problematic and sometimes require a substantial amount of auxiliary coding/modeling to be able to express certain features of the hierarchical model, e.g. causality. On the other hand, UPPAAL offers extra constructs like global and local clocks and time invariants that are not needed for modeling of a hierarchical configuration model.

As can be seen in Table 4, there is an exponential explosion in the number of states in the Freeconf model. The UPPAAL model-checker can handle a large number of states but even for a small Freeconf model like the one presented in Figure 5c, it already consumes over two gigabytes of memory to finish verification of a single query. Using UPPAAL to verify other (possibly) heterogeneous configuration hierarchical models would be problematic. UPPAAL also does not provide a simple way of storing a result of verification (the only report on the overall state of verification is a green or red light next to each query). When one has a functioning and verified model and adds or modifies some aspect of it, one would like to re-verify the model and get all differences in the verification results. One would also like to automatize verification by having a verified reference instance of the model for comparison. None of this is supported in UPPAAL as of yet.

7 Conclusion & Future Works

In this paper, the first step towards a (semi)automatic mechanism of dynamic rules verification has been made. A configuration hierarchical model has been defined and a syntax of propagation rules has been described. The dynamics of evolution of various properties of configuration keys in the configuration tool Freeconf has been briefly introduced as an instance of the general hierarchical model. This Freeconf model instance has been further encoded as a set of state machine automata templates in UPPAAL. The UPPAAL model-checker has been used to verify certain shapes of the Freeconf model.

In the future, a custom domain specific model-checker should be written which would utilize the knowledge of the hierarchical configuration model and the propagation rules and would allow to express a specific problem with minimum overhead. All propagation rules describing the dynamics of the problem should be held at one place for maximum readability. The model-checker should also be able to re-verify the model in the case of an update of the rules. The intention at the moment is to use CHR (Constraint Handling Rules) [Frühwirth, 2009; Frühwirth and Raiser, 2011] as a base programming language for the model-checker implementation.

8 Acknowledgments

This work was partially supported by the project of the Student Grant Agency of the Czech Technical University in Prague No. SGS11/161/OHK4/3T/14, 2011-13.

References

[Alur *et al.*, 1993] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[Arman, 2013] Nabil Arman. Improving rule base quality to enhance production systems performance. *International Journal of Intelligence Science*, 3(1):1–4, 2013.

[Behrmann *et al.*, 2004] Gerd Behrmann, Re David, and Kim G. Larsen. A Tutorial on Uppaal. pages 200–236. Springer, 2004.

[Brand, 2004] Sebastian Brand. *Rule-Based Constraint Propagation Theory and Applications*. PhD thesis, Universiteit van Amsterdam, 2004.

[David *et al.*, 2009] Alexandre David, Tobias Amnellough, and Martin Stiggeore. *Uppaal 4.0: Small Tutorial*, 2009.

[Fabian *et al.*, 2012] David Fabian, Radek Mařík, and Tomáš Oberhuber. Towards a formalism of configuration properties propagation. In *ConfWS'12*, pages 15–20. CEUR Workshop Proceedings, 2012.

[Fabian, 2011] David Fabian. *System for Simplified Generating of Configurations*. Master thesis, Faculty of Nuclear Sciences and Physical Engineering, Prague, 2011. in Czech.

[Fabian, 2012] David Fabian. Freeconf: A general-purpose multiplatform configuration utility. In *Doktorandské dny 2012*, pages 21–30. ČVUT v Praze, 2012.

[Frühwirth and Raiser, 2011] Thom Frühwirth and Frank Raiser. *Constraint Handling Rules: Compilation, Execution, and Analysis*. Books on Demand, 2011.

[Frühwirth, 2009] Thom Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[Liaskos *et al.*, 2005] Sotirios Liaskos, Alexei Lapouchnian, Yiqiao Wang, Yijun Yu, and Steve Easterbrook. Configuring common personal software: a requirements-driven approach. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE '05, pages 9–18, Washington, DC, USA, 2005. IEEE Computer Society.

[Lukichev, 2011] Sergey Lukichev. Improving the quality of rule-based applications using the declarative verification approach. *International Journal of Knowledge Engineering and Data Mining*, 1(3):254–272, December 2011.

[Nardi and Brachman, 2003] Daniele Nardi and Ronald J. Brachman. The description logic handbook. chapter An introduction to description logics, pages 1–40. Cambridge University Press, New York, NY, USA, 2003.

[Preece and Shinghal, 1992] Alun D. Preece and Rajjan Shinghal. Verifying expert systems: A logical framework and a practical tool. *Expert Systems with Applications*, 5:421–436, 1992.

[Preece and Shinghal, 1994] Alun D. Preece and Rajjan Shinghal. Foundation and application of knowledge base verification. *Int J Intell Syst 1994;9(8):683–702. Duftschmid, S. Miksch*, 22:23–41, 1994.

[TechBase, 2012] KDE TechBase. Using KConfig XT. `http://techbase.kde.org/Development/Tutorials/Using_KConfig_XT`, 2012.

[Vlaeminck *et al.*, 2009] Hanne Vlaeminck, Joost Vennekens, and Marc Denecker. A logical framework for configuration software. In *Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*, PPDP '09, pages 141–148, New York, NY, USA, 2009. ACM.

Improving configuration and planning optimization: Towards a two tasks approach

Paul Pitiot[1,2], Michel Aldanondo[1], Elise Vareilles[1], Thierry Coudert[3], Paul Gaborit[1]
[1]University Toulouse – Mines Albi, France
[2] 3IL-CCI Rodez, France
[3]University Toulouse – INP-ENI Tarbes, France
Paul.pitiot@mines-albi.fr

Abstract

This paper deals with mass customization and the association of the product configuration task with the planning of its production process while trying to minimize cost and cycle time. Our research aims at producing methods and constraint based tools to support this kind of difficult and constrained problem. In some previous works, we have considered an approach that combines interactivity and optimization issues and propose a new specific optimization algorithm, CFB-EA (for constraint filtering based evolutionary algorithm). This article concerns an improvement of the optimization step for large problems. Previous experiments have highlighted that CFB-EA is able to find quickly a good approximation of the Pareto Front. This led us to propose to split the optimization step in two sub-steps. First, a "rough" approximation of the Pareto Front is quickly searched and proposed to the user. Then the user indicates the area of the Pareto Front that he is interested in. The problem is filtered in order to restrain the solution space and a second optimization step is done only on the focused area. The goal of the article is to compare thanks to various experimentations the classical single step optimization with the two sub-steps proposed approach.

1 Introduction

This article is about the concurrent optimization of product configuration and production planning. Each problem is considered as a constraint satisfaction problem (CSP) and these two CSP problems are also linked with some constraints. In a previous paper [Pitiot *et al.*, 2013], we have shown that this allows to consider a two-step process: (i) interactive configuration and planning, where non-negotiable user requirements (product requirements and production process requirements) are first processed thanks to constraint filtering and reduce the solution space (ii) optimization of configuration and planning, where negotiable requirements are then used to support the optimization of both product and production process.

Given this problem, product performance, process cycle time and process plus product cost can be optimized, we therefore deal with a multi-criteria problem and our goal is to propose to the user solutions belonging to the Pareto front. For simplicity we only consider cycle time and total cost (product cost plus process cost), consequently the two-step process can be illustrated as shown in figure 1.

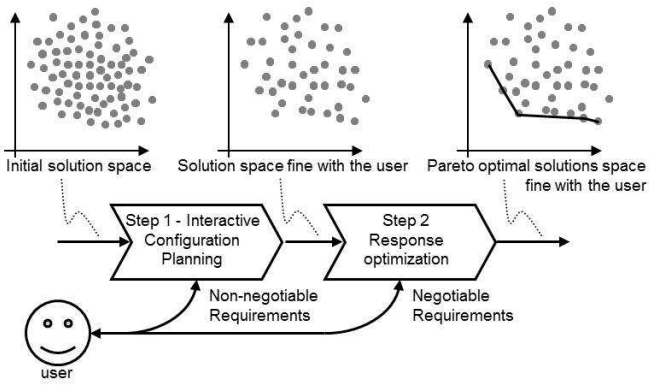


Figure 1 - Two-step process

Some experimental studies, reported last year [Pitiot *et al.*, 2012], discusses optimization performance according to problem characteristics (mainly size and constraint level). That last paper proposes to divide the step 2 (Pareto front computation) in two tasks, particularly in the case of large problems: (i) a first rough computation that permit to have a global idea of possible compromises (ii) a second computation on a restricted area that is selected by the user. The goal of this article is to present experimental results that show that this idea allows to significantly reducing optimization duration while improving optimization quality.

In this introduction, we clarify with a very simple example what we mean by concurrent configuration and planning problem and relevant optimization needs. Then the second section formalizes the optimization problem, presents the optimization algorithm and describes the experimental study. The third section is dedicated to various experimentations.

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

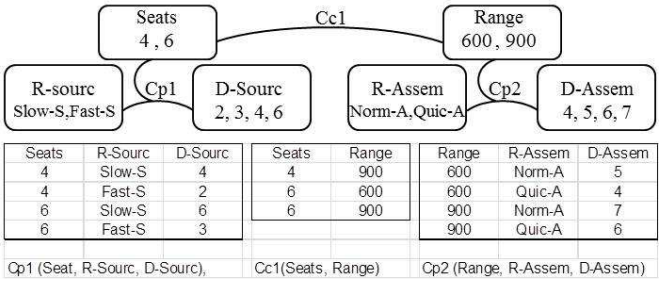
1.1 Configuration and planning processes.

Many authors, since [Mittal and Frayman, 1989], [Soininen *et al.*, 1998] or [Aldanondo *et al.*, 2008] have defined configuration as the task of deriving the definition of a specific or customized product (through a set of properties, sub-assemblies or bill of materials, etc…) from a generic product or a product family, while taking into account specific customer requirements. Some authors, like [Schierholt 2001], [Bartak *et al.*, 2010] or [Zhang *et al.* 2013] have shown that the same kind of reasoning process can be considered for production process planning. They therefore consider that deriving a specific production plan (operations, resources to be used, etc...) from some kind of generic process plan while respecting product characteristics and customer requirements, can define production planning. Many configuration and planning studies (see for example [Junker, 2006] or [Laborie, 2003]) have shown that each problem could be successfully considered as a constraint satisfaction problem (CSP). We proposed to associate them in a single CSP in order to process them concurrently.

This concurrent process and the supporting constraint framework present three main interests. First they allow considering constraints that links configuration and planning in both directions (for example: a luxury product finish requires additional manufacturing time or a given assembly duration forbids the use of a particular kind of component). Secondly they allow processing planning requirements even if product configuration is not completely defined, and therefore avoid the traditional sequence: configure product then plan its production. Thirdly, CSP fit very well on one side, interactive process thanks to constraint filtering techniques, and on the other side, optimization thanks to various problem-solving techniques. However, we assume infinite capacity planning and consider that production is launched according to each customer order and production capacity is adapted accordingly.

In order to illustrate the problem to solve we recall the very simple example, proposed in [Pitiot *et al.*, 2012], dealing with the configuration and planning of a small plane. The constraint model is shown in figure 2. The plane is defined by two product variables: number of seats (Seats, possible values 4 or 6) and flight range (Range, possible values 600 or 900 kms). A configuration constraint Cc1 forbids a plane with 4 seats and a range of 600 kms. The production process contains two operations: sourcing and assembling. (noted Sourc and Assem). Each operation is described by two process variables: resource and duration: for sourcing, the resource (R-Sourc, possible resources “Fast-S” and “Slow-S”) and duration (D-Sourc, possible values 2, 3, 4, 6 weeks), for assembling, the resource (R-Assem, possible resources “Quic-A” and “Norm-A”) and duration (D-Assem, possible values 4, 5, 6, 7 weeks).

Two process constraints linking product and process variables modulate configuration and planning possibilities: one linking seats with sourcing, Cp1 (Seat, R-Sourc, D-Sourc), and a second one linking range with the assembling, Cp2 (Range, R-Assem, D-Assem). The allowed combinations of each constraint are shown in the 3 tables of figure 2 and lead to 12 solutions for both product and production process.



Seats	R-Sourc	D-Sourc
4	Slow-S	4
4	Fast-S	2
6	Slow-S	6
6	Fast-S	3

Cp1 (Seat, R-Sourc, D-Sourc),

Seats	Range
4	900
6	600
6	900

Cc1(Seats, Range)

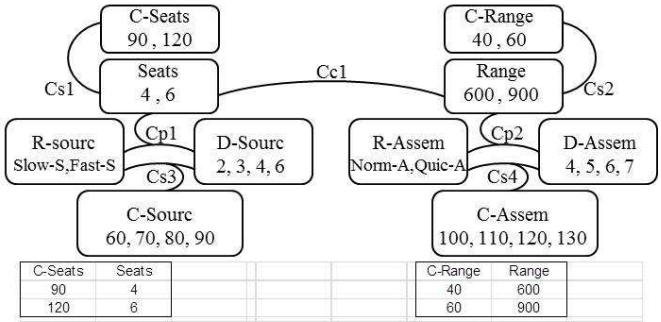
Range	R-Assem	D-Assem
600	Norm-A	5
600	Quic-A	4
900	Norm-A	7
900	Quic-A	6

Cp2 (Range, R-Assem, D-Assem)

Figure 2 - Concurrent configuration and planning CSP model

1.2 Optimization needs

With respect to the previous problem, once the customer or the user has provided his non-negotiable requirements, he is frequently interested in knowing what he can get in terms of price and delivery dates (performance is not considered any more). Consequently, the previous model must be updated with some variables and numerical constraints in order to compute the two criteria. The cycle time matches the ending date of the last production operation of the configured product. Cost is the sum of the product cost and process cost.



C-Seats	Seats
90	4
120	6

Cs1 (C-Seats, Seats)

C-Range	Range
40	600
60	900

Cs2 (C-Range, Range)

C-Sourc	R-Sourc	D-Sourc
60	Slow-S	4
80	Fast-S	2
70	Slow-S	6
90	Fast-S	3

Cs3 (C-Sourc, R-Sourc, D-Sourc),

C-Assem	R-Assem	D-Assem
100	Norm-A	5
120	Quic-A	4
110	Norm-A	7
130	Quic-A	6

Cs4 (C-Assem, R-Assem, D-Assem)

Figure 3 - CSP model to optimize

The model of figure 2 is completed in figure 3. For cost, each product variable and each process operation is associated with a cost parameter and a relevant cost constraint: (C-Seats, Cs1), (C-Range, Cs2), (C-Sourc, Cs3) and (C-Assem, cs4) detailed in the tables of figure 3.

The total cost and cycle time are obtained with a numerical constraint as follows:

Total cost = C-Seats + C-Range + C-Sourc + C-Assem.

Cycle time = D-Sourc + D-Assem

The twelve previous solutions are shown on the figure 4 with the Pareto front gathering the optimal ones. The goal of this article is to improve the computation of this Pareto front with respect to the user preference.

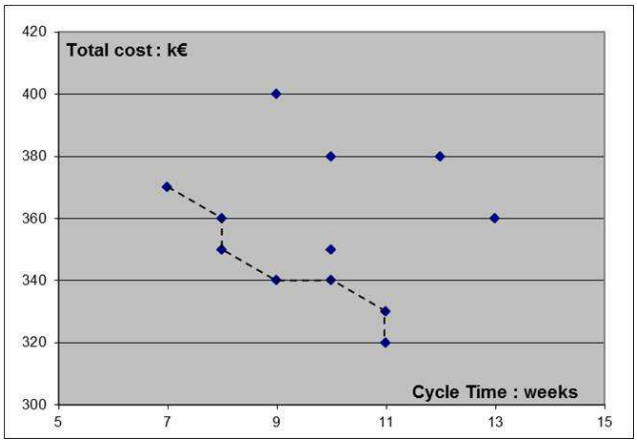


Figure 4 – Problem solutions and Pareto front

2 Optimization problem and techniques

The optimization problem is first defined, and then the optimization algorithm that will be used is described. Finally, the experimental process is introduced.

2.1 Optimization problem

The optimization problem can be generalized as the one shown in figure 5.

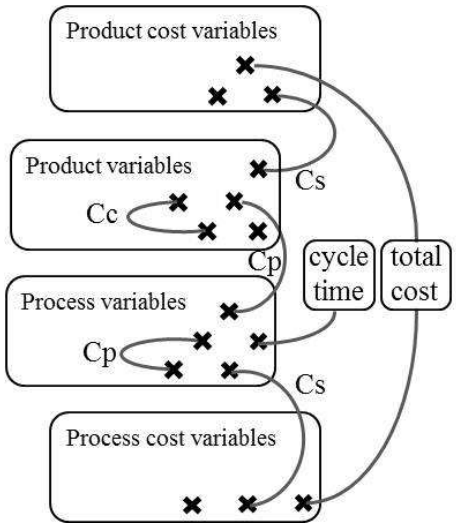


Figure 5 – Constrained optimization problem

The constrained optimization problem (O-CSP) is defined by the quadruplet <V, D, C, f > where V is the set of decision variables, D the set of domains linked to the variables of V, C the set of constraints on variables of V and f the multi-valued fitness function. The set V gathers: the product variables and the resource process variables (we assume that duration process variables are deduced from product and resource). The set C gathers: only configuration constraints (Cc) and process constraints (Cp). The variables operation durations and cycle time are linked with a numerical constraint that does not impact solution definition and therefore does not belong to V and C. The same applies to the product/process cost variables and total cost, which are linked with cost constraints (Cs) and total cost constraints. The filtering system allows dynamically updating the domain of all these variables with respect to the constraints. The variables belonging to V are all symbolic or at least discrete. Duration and cost variables are numerical and continuous. Therefore, constraints are discrete (Cc), numerical (cycle time and total cost) or mixed (Cp and Cs). Discrete constraints filtering is processed using a conventional arc consistency technique [Bessiere, 2006] while numerical constraints are processed using bound consistency [Lhomme, 1993].

2.2 Optimization algorithm

A strong specificity of this kind of optimization problem is that the solution space is large. [Amilhastre et al, 2002] report that a configuration solution space of more than $1.4*10^{12}$ is required for a car-configuration problem. When planning is added, the combinatorial structure can become huge. Another specificity lies in the fact that the shape of the solution space is not continuous and, in most cases, shows many singularities. Furthermore, the multi-criteria problem and the need for Pareto optimal results are also strong problem expectations. These points explain why most of the articles published on this subject, as for example [Hong *et al.*, 2010] or [Li *et al.*, 2006] consider genetic or evolutionary approaches to deal with this problem. In this article we will use "CFB-EA" (for Constraint Filtering Based Evolutionary Algorithm) a promising algorithm that we have designed specifically for this problem.

CFB-EA is based on the SPEA2 method [Zitzler *et al.*, 2001] which is one of the most useful Pareto-based methods. It's based on the preservation of a selection of best solutions in a separate archive. It includes a performing evaluation strategy that brings a well-balanced population density on each area of the search space, and it uses an archive truncation process that preserves boundary solution. It ensures both a good convergence speed and a fair preservation of solutions diversity.

To deal with constrained problems, we completed this method with specific evolutionary operators (initialization, uniform mutation and uniform crossover) that preserve feasibility of generated solutions.

This provides the six steps following approach:

1. Initialization of individual set that respect the constraints (thanks to filtering),
2. Fitness assignment (balance of Pareto dominance and solution density)
3. Individuals selection and archive update
4. Stopping criterion test
5. Individuals selection for crossover and mutation operators (binary tournaments)
6. Individuals crossover and mutation that respect the constraints (thanks to filtering)
7. Return to step 2.

For initialization, crossover and mutation operators, each time an individual is created or modified, every gene (decision variable of V) is randomly instantiated into its current domain. To avoid the generation of unfeasible individuals, the domain of every remaining gene is updated by constraint filtering. As filtering is not full proof, inconsistent individuals can be generated. In this case a limited backtrack process is launched to solve the problem. This approach doesn't need any additional parameter tuning for constraint handling. In the following, we will briefly remind the principles and operators used in CFB-EA.

Many research studies try to integrate constraints in EA. C. Coello Coello proposes a synthetic overview in [Mezura-Montes and Coello Coello 2011]. The current tendencies in the resolution of constrained optimization problem using EAs are penalty functions, stochastic ranking, ε-constrained, multi-objective concepts, feasibility rules and special operators. CFB-EA belongs to this last family.

The special operators class gathers methods that try to deal only with feasible individuals like repairing methods, preservation of feasibility methods or operator that move solutions within a specific region of interest within the search space as for example the boundaries of the feasible region. Generally and has we verified on our last experimentations, these methods are known to be performing on non-over-constrained problems (i.e. a feasible solution can be obtained in a reasonable amount of time to be able to generate a population of solutions).

CFB-EA aims at preserving the feasibility of the individuals during their construction or modification. Proposed specific evolutionary operators prune search space using constraint filtering. The main difference between our approach and others is that we do not have any infeasible solution in our population or archive. Each time we modify an individual, the constraints filtering system is used in order to verify consistency preservation of individuals.

Previous experimentations [Pitiot *et al.*, 2012] allowed us to verify that the exact approaches are limited to problems of limited size and that CFB-EA is completely competitive for the level of constraint of the models which interest us. In this article, we propose a new two sub-step optimization approach that takes advantage of the three following characteristics: (i) EA are anytime algorithms, e.g. they can supply a set of solutions (Pareto Front) at any time after initialization, (ii) we have an user who can possibly refine his criteria requirements with regard to the solutions obtained during optimization process ; (iii) CFB-EA is relevant for the range of concurrent configuration and planning problems required (size and constraints level) and more particularly it can propose, in a reasonable amount of time, a good approximation of the Pareto Front that allows the user to decide about his own cost/cycle time compromise.

2.3 Two-task optimization approach.

As explained in the introduction, the goal of this article is to evaluate, for large problem, the interest of replacing the single shot Pareto front computation by two successive tasks: (i) a first rough computation that provides a global idea of possible compromises (ii) a second computation on a restricted area selected by the user.

This is shown in the illustration of figure 6. The left part of figure 6 shows a single shot Pareto. The right part of figure 6 shows a rough Pareto quickly obtained (first task), followed by a zoom selected by the user (max cost and max time) and a second Pareto computation only on this restricted area (second task). The restricted area is obtained by constraining the two criteria total cost and cycle time (or interesting area) and filtering these reductions on the whole problem.

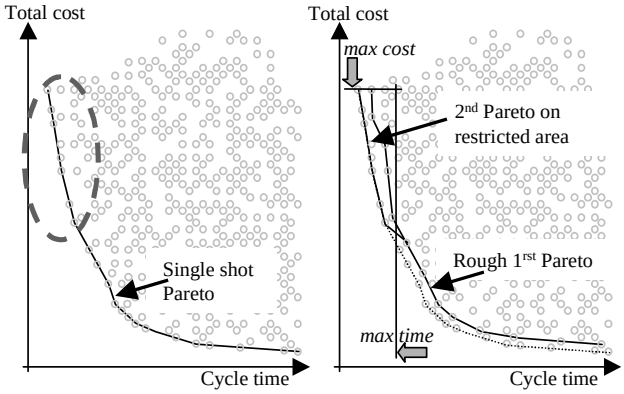


Figure 6 – Single shot and two-task optimization principles

The second optimization task does not restart from scratch. It benefits from the individuals of the archive that belongs to the restrained area founded during first task. We thus replaced the initialization of our CFB-EA (constitution of the first population) by a selection of a set of the best solutions obtained during the first rough optimization.

This provides the following process:

1. Interactive configuration and planning using non-negotiable requirements of the user (as before),

2.1 - 1st global optimization task on negotiable requirements of the user

2.2 - 2nd optimization on interesting area initialized with individuals of the previous step.

3 Experimentations

3.1 Model used and performance measure

The goal of the proposed experiments is to compare these two optimization approaches (single-shot and two-task optimization approaches) in terms of result quality and computation time. In terms of quality we want to compare the two fronts and will use the Hypervolume measurement proposed by [Zitzler and Thiele 1998] which is illustrated in figure 7. It measures the hypervolume of the space dominated by a set of solutions. It thus allows evaluating both convergence and diversity proprieties (the fittest and most diversified set of solutions is the one that maximizes hypervolume). In terms of computation time, we want to evaluate, for a given Hypervolume result the time reduction provided by the second approach.

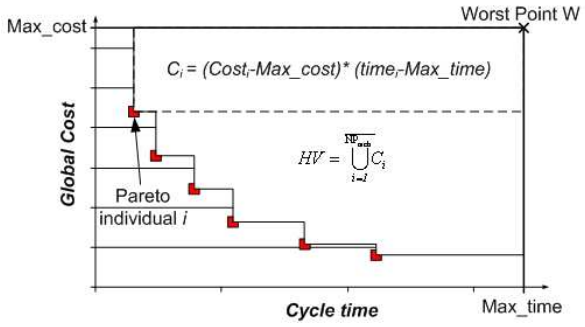


Figure 7 – Hyper volume definition

In terms of problem size, we consider a model called "full_ aircraft" that gathers 92 variables (symbolic, integer or float variables) linked by 67 constraints (compatibility tables, equations or inequalities). Among these variables, we find 21 decision variables that will be manipulated by the optimization algorithms (chromosome in EAs):

- 12 variables (each with 6 possible discrete values) that describe product customization possibilities,
- 9 variables (each with 9 possible discrete values) that describe production process possibilities. In fact, the nine values aggregate 3 resource types and 3 resource quantities for each of the 9 process operations that compose the production process.

Without any constraints, this provides a number of possible combinations around 10^{18} ($\approx 6^{12} \times 9^9$). An average constraint level (around 93% of solutions rejected) allows $7.3*10^{16}$ feasible solutions. Results of experimentation's with other model sizes and other constraint levels can be consulted in [Pitiot *et al.*, 2012].

Figure 8 shows the Pareto Fronts obtained with CFB-EA after 3 and 24 hours of computation. The rough Pareto front obtained after 3 hours of computation allows the user to decide in which area he is interested in. In the next subsection, we will study a division of this Pareto front in three restricted area:

- Aircraft_zoom_1: area that correspond to solutions with a cycle time less than 410 (solutions with shortest cycle times),
- Aircraft_zoom_2: area that correspond to solutions with a cycle time less than 470 and a total cost less than 535 (compromise solutions),
- Aircraft_zoom_3: area that correspond to solutions with a total cost less than 475 (solutions with lowest total costs).

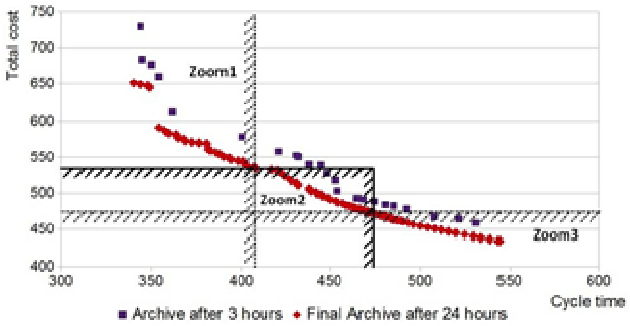


Figure 8 –Pareto-fronts obtained on "full aircraft model" after 3 and 24 hours of computation

These three areas correspond with a division of the final Pareto front obtained after 24h of computation in three equal parts. These areas have been selected in order to evaluate performance of the proposed two-task approach, but it also corresponds with some classical preference of a user who could wish: (i) a less expensive plane, (ii) a short cycle time, (iii) a compromise between total cost and cycle time. We will discuss this aspect in section 3.3.
The optimization algorithms were implemented in C++ programming language and interacted with the filtering system coded in Perl language. All tests were done using a laptop computer powered by an Intel core i5 CPU (2.27 Ghz, only one CPU core is used) and using 2.8 Go of ram.

3.2 Two-task approach evolutionary settings

For a first experimentation of the two-task approach, we use classical evolutionary settings (e.g. the same evolutionary settings used for the single-shot approach: Population size: 80, Archive size: 100, Individual Mutation Probability: 0.3, Gene Mutation Probability: 0.2, Crossover Probability: 0.8). The main difference with the single-shot approach is with the backtrack limit (e.g. number of allowed backtrack in mutation or crossover operator). This limit has been set to 100 in the one-shot approach and to 30 in the two-step approach.
Indeed in the two-step approach, it could be time consuming to obtain a valid solution. For example with the single-shot optimization, only 2.5% of filtered individuals were unfeasible and none of them were abandoned; while with the two-task approach and a lower backtrack limit, around 7% of filtered individuals were unfeasible and 0.3% of them were abandoned. So a lower backtrack limit reduces the time spend to try to repair unfeasible individuals.

The only other difference between single-shot CFB-EA and two-task CFB-EA is the stopping criterion. While in single-shot approach, we use a fix time limit (24hours), the two-task approach uses a bcondition stopping test that stops either if there is no HV improvement after 2 hours or after 12 hours of computation (that must be added to the three initial hours for getting the rough Pareto Front).

3.3 Experimental results

The goal of this section is to evaluate the two-task optimization on the three selected areas of figure 8 (zoom 1, zoom 2 and zoom3) with respect to the single-shot optimization.

First result illustrations

Figure 9 illustrates an example of the Pareto fronts that can be obtained on the zoom 1 area :
- rough Pareto obtained after 3 hours (fig 9 squares),
- two-task, after 3+12 hours (fig 9 triangles),
- single-shot, stopped after 24 hours (fig 9 diamonds).

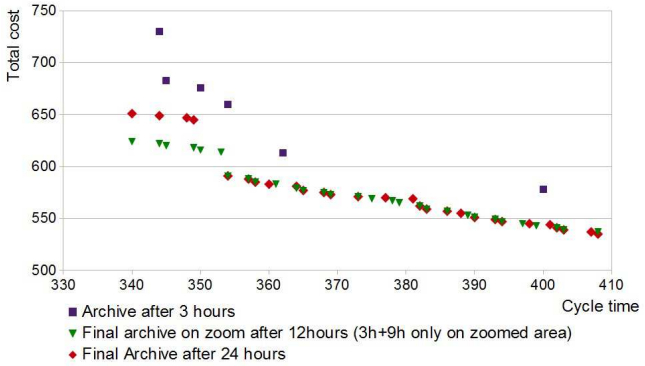


Figure 9 –Example of Pareto fronts obtained on zoom1

The Pareto Fronts obtained by the two approaches (single-shot and two-task) are very close when the cycle is greater than 355. For lower cycle times, the proposed two-task approach is a little better. However, these curves correspond with a specific run. In order to derive stronger conclusions, 10 executions of the two approaches have been achieved for each of the three zoom areas.

Detailed comparisons

Detailed experimental results achieved on the three zoom areas are presented in figure 10 and table 1.
On each graph of figure, the vertical axis corresponds to the hyper volume (average of ten runs) reach and horizontal one is the time spent. At time 0, the single-shot optimization is launched (dotted line). After 3 hours (10800 seconds):
- the single-shot keeps going on (dotted line),
- the two-task is launched (solid line).

The table provides numeric results for each zoom area. The columns display the single-shot, two-task and % gap of:
- average final hypervolume,
- average % standard deviation of hypervolume
- average computation time,
- average % standard deviation of computation time,
- maximum value of hypervolume.

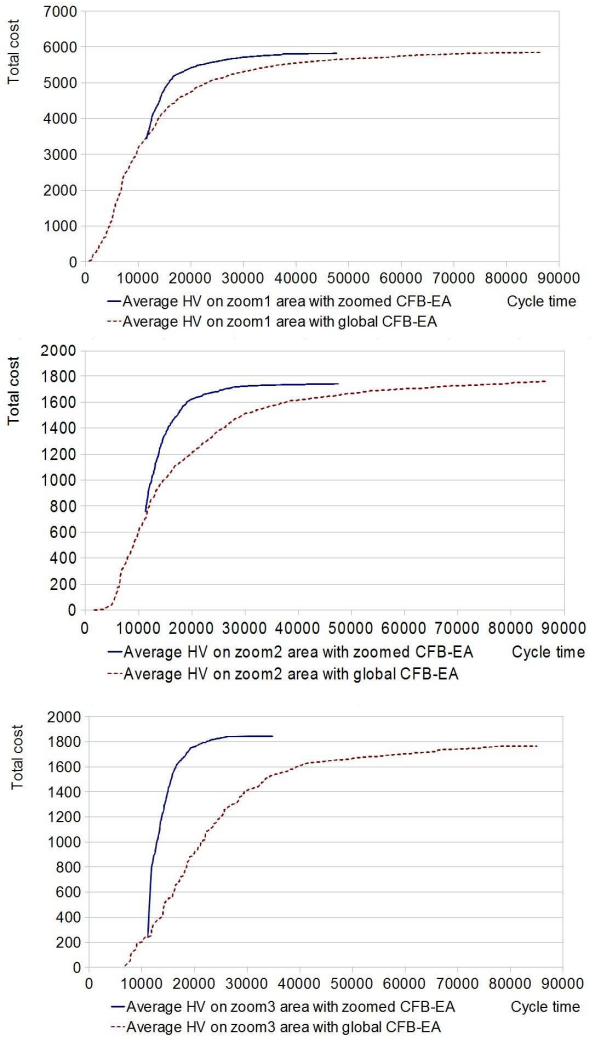


Figure 10 – Evolution of hypervolume

In terms of quality, the new proposed approach (two-task optimization) allows to obtain a similar performance with respect to single-shot one:
- 0.4% worse on zoom1
- 1% worse on zoom2
- 4% better on zoom3

but in around half of computing time:
- 13 h instead of 24h for on zoom1
- 13.5h instead of 24h for on zoom2
- 10.5h instead of 24h for on zoom 3.

Furthermore, this computing time includes the 2 hours of computation without any hypervolume reduction before stopping (stopping criterion of the two-task approach).

It can be seen on the figure10 that when the single-shot CFB-EA has trouble to obtain a good Pareto Front during the first three hours, the more the two-task CFB-EA is performing. On zoom1 area, single-shot CFB-EA reaches relatively quickly a near-final Pareto Front; while on zoom3 area, it reaches it very slowly.

		Single-shot CFBEA	Two-task CFBEA	gap in %
Zoom1	Average Final HV	5849	5823	-0.4
	Average HV RSD	3.8%	5.1%	
	Total time	86400(24h)	47996 (≈13h)	-44.6
	Total time RSD	0	15%	
	Max HV	6043	6057	0.2
Zoom2		Single-shot CFBEA	Two-task CFBEA	gap in %
	Average Final HV	1758	1740	-1.
	Average HV RSD	2.1%	2.3%	
	Total time	86400(24h)	48501 (≈13.5h)	-44
	Total time RSD	0	16%	
	Max HV	1795	1776	-1
Zoom3		Single-shot CFBEA	Two-task CFBEA	gap in %
	Average Final HV	1765	1844	4.4
	Average HV RSD	3.16%	0.07%	
	Total time	86400(24h)	38185 (≈10.5h)	-55.9
	Total time RSD	0	26%	
	Max HV	1831	1845	0,7

Table 1. Comparison of the two approaches

4 Conclusions

The goal of this paper was to evaluate a new optimization principle that can handle concurrent configuration and planning. First the background of concurrent configuration and planning has been recalled with associated constrained modeling elements. Then an initial optimization approach (single-shot CFB-EA) was described followed by the two-task approach object of this paper.

Instead of computing a Pareto Front on the whole solution space, the key idea is: to compute quickly a rough Pareto Front, to ask the user about an interesting area and, to launch Pareto computation only on this area.

According to experimental results, in terms of computation time, the new two-task approach allows a significant time saving around half of the previous time needed by the single-shot optimization approach. In terms of quality, Hypervolume computation are very close or even a little better in some case.

Furthermore, these results have been obtained on a rather large problem that contains around $10^{16}/10^{17}$ solutions. With smaller problems, the proposed approach should perform much better. We are already working on a more extensive test (different model size and different level of constraints) as we did in [Pitiot *et al.*, 2012]. Another key aspect that needs to be study is to find a way to define the rough Pareto computation time.

References

[Aldanondo *et al.*, 2008] M. Aldanondo, E. Vareilles. Configuration for mass customization: how to extend product configuration towards requirements and process configuration, Journal of Intelligent Manufacturing, vol. 19 n° 5, p. 521-535A (2008)

[Amilhastre et al, 2002] J. Amilhastre, H. Fargier, P. Marquis, Consistency restoration and explanations in dynamic csps - application to configuration, in: Artificial Intelligence vol.135, 2002, pp. 199-234.

[Bartak *et al.*, 2010] R. Barták, M. Salido, F. Rossi. Constraint satisfaction techniques in planning and scheduling, in: Journal of Intelligent Manufacturing, vol. 21, n°1, p. 5-15 (2010)

[Bessiere, 2006] C. Bessiere, Handbook of Constraint Programming, Eds. Elsevier, chap. 3 Constraint propagation, 2006, pp. 29-70.

[Hong *et al.*, 2010] G. Hong, D. Xue, Y. Tu,, Rapid identification of the optimal product configuration and its parameters based on customer-centric product modeling for one-of-a-kind production, in: Computers in Industry Vol.61 n°3, 2010, pp. 270–279.

[Junker, 2006] U. Junker. Handbook of Constraint Programming, Elsevier, chap. 24 Configuration, p. 835-875 (2006)

[Laborie, 2003] P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results, in: Artificial Intelligence vol 143, 2003, pp 151-188.

[Lhomme, 1993] O. Lhomme. Consistency techniques for numerical CSPs, in: proc. of IJCAI 1993, pp. 232-238.

[Li *et al.*, 2006] L. Li, L. Chen, Z. Huang, Y. Zhong, Product configuration optimization using a multiobjective GA, in: I.J. of Adv. Manufacturing Technology vol. 30, 2006, pp. 20-29.

[Mezura-Montes and Coello Coello 2011] E. Mezura-Montes, C. Coello Coello, Constraint-Handling in Nature-Inspired Numerical Optimization: Past, Present and Future, in: Swarm and Evolutionary Computation, Vol. 1 n°4, 2011, pp. 173-194

[Mittal and Frayman, 1989] S. Mittal, F. Frayman. Towards a generic model of configuration tasks, proc of IJCAI, p. 1395-1401(1989).

[Pitiot *et al.*, 2012] P. Pitiot, M. Aldanondo, E. Vareilles, L. Zhang, T. Coudert. Some Experimental Results Relevant to the Optimization of Configuration and Planning Problems, in : Lecture Notes in Computer Science Volume 7661, 2012, pp 301-310

[Pitiot *et al.*, 2013] P. Pitiot, M. Aldanondo, E. Vareilles, P. Gaborit, M. Djefel, S. Carbonnel, Concurrent product configuration and process planning, towards an approach combining interactivity and optimality, in: I.J. of Production Research Vol. 51 n°2, 2013 , pp. 524-541.

[Schierholt 2001] K. Schierholt. Process configuration: combining the principles of product configuration and process planning AI EDAM / Volume 15 / Issue 05 / novembre 2001 , pp 411-424

[Soininen *et al.*, 1998] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, Towards a General Ontology of Configuration., in: Artificial Intelligence for Engineering Design, Analysis and Manufacturing vol 12 n°4, 1998, pp. 357–372.

[Zhang *et al.*, 2013] L. Zhang, E. Vareilles, M. Aldanondo. Generic bill of functions, materials, and operations for SAP2 configuration, in: I.J. of Production Research Vol. 51 n°2, 2013 , pp. 465-478.

[Zitzler and Thiele 1998] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms - a comparative case study, in: proc. of 5th Int. Conf. on parallel problem solving from nature, Eds. Springer Verlag, 1998, pp. 292-301.

[Zitzler *et al.*, 2001] E. Zitzler, M. Laumanns, L. Thiele., SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report 103, Swiss Fed. Inst. of Technology (ETH), Zurich (2001)

Recommender Systems for Configuration Knowledge Engineering *

A. Felfernig, S. Reiterer, M. Stettinger, F. Reinfrank, M. Jeran, and G. Ninaus
Graz University of Technology
Inffeldgasse 16b, A-8010 Graz, Austria
{felfernig,reiterer,stettinger,reinfrank,jeran,ninaus}@ist.tugraz.at

Abstract

The knowledge engineering bottleneck is still a major challenge in configurator projects. In this paper we show how recommender systems can support knowledge base development and maintenance processes. We discuss a couple of scenarios for the application of recommender systems in knowledge engineering and report the results of empirical studies which show the importance of user-centered configuration knowledge organization.

1 Introduction

Product knowledge changes frequently [Soloway, 1987]. Therefore, it must be possible to conduct knowledge base development and maintenance operations efficiently. Since the early developments of configurator applications in the late 1970's and early 1980's [McDermott, 1982], knowledge representations have been improved in terms of (1) *model-based approaches* which allow a clear separation of domain knowledge and problem solving algorithms, (2) *higher-level knowledge representations* which allow a component-oriented representation of configuration knowledge (see, e.g., [Stumptner *et al.*, 1998]), and (3) *graphical knowledge representations* (e.g., [Felfernig *et al.*, 2000; 2001]) which allow a compact representation. In addition to new knowledge representations, intelligent *diagnosis approaches* have been developed which help a knowledge engineer to identify and repair erroneous configuration knowledge [Junker, 2004; Felfernig *et al.*, 2004; 2009; 2013].

Due to diversification strategies of companies, product and service assortments are becoming increasingly large and complex [Huffman and Kahn, 1998]. The complexity of the underlying knowledge bases increases to the same extent which requires additional concepts that help a knowledge engineer to conduct knowledge base development and maintenance operations in an efficient fashion. Furthermore, knowledge bases are often developed by a group of persons with different knowledge, goals, and focuses with regard to development and maintenance operations. This situation requires *adaptive user interfaces* to be integrated into configuration knowledge engineering environments. Adaptive user interfaces for knowledge engineering have the potential to effectively support engineers and domain experts in activities such as *learning* (knowledge base understanding), *finding* (the relevant items in the knowledge base), and *testing & debugging* (removing the source of faulty behavior).

In order to offer more adaptivity in configurator development environments, we propose the application of different types of recommendation technologies [Jannach *et al.*, 2010] which proactively support domain experts and engineers when creating and adapting configuration knowledge. Such technologies should dispose of a basic understanding of cognitive processes when persons develop and maintain configuration knowledge bases. They should support functionalities such as *recommending* relevant items (variables, component types, constraints, diagnoses, etc.) and simultaneously *omitting* specific items that are not relevant. Recommender systems have the potential to provide such a support (see, e.g., [Robillard *et al.*, 2010]).

There are three basic recommendation approaches. *First*, *collaborative filtering* [Konstan *et al.*, 1997] determines recommendations based on the preferences of *nearest neighbors* (users with similar preferences compared to the current user). In this context, items are recommended to the current user which have received a positive rating by the nearest neighbors but are not known to the current user. *Second*, *content-based filtering* [Pazzani and Billsus, 1997] recommends items that are not known to the current user and are similar to items that have already been purchased by her/him. Similarity between items can be determined, for example, on the basis of the similarity of keywords used to describe the item. *Third*, *knowledge-based recommenders* recommend items by using constraints or similarity metrics [Burke, 2000; Felfernig and Burke, 2008].

This paper is organized as follows. In Section 2 we introduce example scenarios for the application of recommender technologies in knowledge engineering. Thereafter, we report results of related empirical studies (see Section 3). In Section 4 we provide a discussion of related work. Conclusions and a discussion of future research issues are given in Section 5.

2 Recommenders for Knowledge Engineering

Collaborative Recommendation of Constraints. Collaborative filtering (CF) recommender systems have shown to be

*The work presented in this paper has been funded by the Austrian Research Promotion Agency (Project: ICONE (827587)).

one of the best choices to achieve *serendipity effects*, i.e., to be surprised (in a positive sense) by item recommendations one did not expect when starting the recommendation process. In situations were knowledge engineers do not know the configuration knowledge base very well, collaborative recommendations can be exploited to support a more focused analysis of the knowledge base. The availability of navigation data from other knowledge engineers is the major precondition for determining recommendations with collaborative filtering. Table 1 shows an example of navigation data that describes in which order knowledge engineers (users) accessed the constraints of a knowledge base. For simplicity we assume that each of the users accessed each constraint (but in different order). Similar applications of collaborative filtering can be imagined for the recommendation of variables (or component types) and instances of a component catalog.

Table 1 stores the information in which order the constraints have been visited by knowledge engineers (users), for example, user 1 analyzed the constraints in the order $[c_5, c_2, c_3, c_1, c_4, c_6]$. Let us assume that the current user has already visited the constraints c_5 and (then) c_2. The nearest neighbors of the current user (users with a similar navigation behavior) are the users 1, 2, and 4. The majority of these users analyzed constraint c_1 in the third step – this one will be recommended to the current user. Note that this recommendation approach is currently under evaluation, therefore no related empirical results will be reported in Section 3.

user	c_1	c_2	c_3	c_4	c_5	c_6
1	4	2	3	5	1	6
2	3	2	5	6	1	4
3	1	3	2	4	6	5
4	3	2	4	5	1	6
current	?	2	?	?	1	?

Table 1: Recommending constraints (c_i) with *CF*.

Content-based Clustering of Constraints. Another possibility to support knowledge engineers is to cluster constraints with the goal to improve the overall clarity of the knowledge base. We will exemplify this on the basis of *k-means clustering* [Witten and Frank, 2005]. Following this approach, we have to generate k initial *centroids* which act as (first) representatives of future clusters. In the following, each object (in our case: constraint) is assigned to the group (cluster) with the closest (most similar) centroid. Thereafter, centroids are recalculated. In our case, a centroid is defined as the object with the highest overall similarity to the other objects in the cluster. The algorithm terminates if the centroids are *stable* (do not change). *k-means clustering* is guaranteed to terminate but is not necessarily optimal since the outcome depends on the initial centroids ([Witten and Frank, 2005]).

For demonstration purposes we introduce the following simple configuration problem which is represented as a basic constraint satisfaction problem (CSP = (V, D, C)) where V represents a set of variables $\{v_1, v_2, ..., v_5\}$, D represents the set of corresponding domains ($dom(v_i) = \{1..5\}$), and C represents the following set of constraints.

$\{c_1 : v_1 = 3 \rightarrow v_2 > 1, c_2 : v_1 = 3 \wedge v_3 = 1, c_3 : v_2 = 2 \rightarrow v_3 = 1, c_4 : v_3 = 1 \rightarrow v_1 \neq 1, c_5 : v_3 = 1 \rightarrow (v_4 = 2 \wedge v_1 > v_5), c_6 : v_4 \geq 1 \rightarrow v_5 \leq 4, c_7 : v_5 = 1 \rightarrow v_3 = 2 \vee v_3 = 3\}$.

On the basis of this simple knowledge base, we can calculate the similarities between the individual constraints (c_a, c_b) by using Formula 1. In this formula, $V = variables(c_a) \cup variables(c_b)$, $co{-}occurrence(v, c_a, c_b)$ = 1 if v is contained in both constraints on the same position, $co{-}occurrence(v, c_a, c_b)$ = 0.5 if v is contained in both constraints but on a different position, and $co{-}occurrence(v, c_a, c_b)$ = 0 of no co-occurrence exists. Note that this is *one possible approach to similarity determination*. We also compared this approach with operator-based similarity and a random assignment of constraints to clusters.

$$sim(c_a, c_b) = \frac{\sum_{v \in V} co\text{–}occurrence(v, c_a, c_b)}{|V|} \quad (1)$$

The similarities between the pairs of individual constraints are depicted in Table 2.

$c_i \in C$	c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_1	1.0	-	-	-	-	-	-
c_2	0.33	1.0	-	-	-	-	-
c_3	0.16	0.33	1.0	-	-	-	-
c_4	0.16	0.5	0.16	1.0	-	-	-
c_5	0.1	0.25	0.1	0.37	1.0	-	-
c_6	0.0	0.0	0.0	0.0	0.12	1.0	-
c_7	0.0	0.33	0.33	0.16	0.12	0.16	1.0

Table 2: Similarities between individual constraints.

On the basis of these individual similarities we are able to determine a set of corresponding clusters ($k = 2$). The determination of such clusters is exemplified in Table 3. First, we (randomly) select two constraints as initial cluster centers (centroids): c_1 and c_5 (denoted by cs). In iteration 2 the center of cluster 1 changes to c_2 and we have to re-calculate the cluster assignment. After this iteration, the assignment is stable, i.e., the cluster centers (c_2 and c_5) remain the same.

iteration	c_1	c_2	c_3	c_4	c_5	c_6	c_7
1	$1(cs)$	1	1	2	$2(cs)$	2	2
2	1	$1(cs)$	1	1	$2(cs)$	2	1

Table 3: k-means clustering of $C = \{c_1, c_2, ..., c_7\}$.

For the visualization of the constraints $\{c_1, c_2, ..., c_7\}$ this means that the knowledge base would be presented in terms of two constraint groups: $\{c_1, c_2, c_3, c_4, c_7\}$ and $\{c_5, c_6\}$.

Knowledge-based Refactoring Recommendations. The way in which semantics is expressed has an impact on the understandability of the knowledge base. For example, users need less time to understand the semantics of a knowledge base if implications are expressed in terms of $A \rightarrow B$ compared to the alternative representation of $\neg A \vee B$. Explicit knowledge about the cognitive complexity of constraint representations can be exploited to recommend structural and

semantics-preserving adaptations of knowledge structures. Such recommendations are knowledge-based, since they are explicitly encoded in refactoring rules.

3 Empirical Evaluation

For the *content-based clustering of constraints* and *knowledge-based refactoring recommendations* we now present the results of two empirical studies. In the first study, we compared the applicability of three different clustering strategies with regard to knowledge engineering tasks (*find a solution*, *find a minimal conflict*) (see, e.g., [Junker, 2004]).

Study A: Clustering of Constraints. For two different configuration knowledge bases (kba_1, kba_2) we conducted a study based on an within-subjects design (N=40). Each study participant (students of computer science who visited a related course on knowledge engineering) had the task of (1) finding a solution (in kba_1) and (2) finding a minimal conflict (in kba_2).[1] There were no time limits regarding task completion. Each student was assigned to one type of clustering (one out of variable-based similarity, operator-based similarity, and random clustering), i.e., we did not vary the type of clustering per student. The knowledge bases (kba_1, kba_2) were defined as CSPs in a domain-independent fashion in order to avoid an additional cognitive complexity related to the understanding of a product domain. The basic properties of the used knowledge bases are summarized in Table 4.

Knowledge base	#($v_i \in V$)	v_i domain size	#($c_i \in C$)
kba_1	5	5	15
kba_2	10	3	10

Table 4: Knowledge bases used in *Study A*.

The outcome of this experiment is shown in Table 5.

Grouping approach	kba_1*: SOL*	kba_2*: CON*
Similar variables	21.43%	42.86%
Similar operators	30.77%	53.85%
Random	38.46%	76.92%

Table 5: Error rates for completing the tasks *find a solution (SOL)* and *find a conflict (CON)* depending on clustering approach (variable-based, operator-based, or random).

From the three compared approaches to the clustering of constraints in a configuration knowledge base, variable similarity based clustering clearly outperforms operator-based clustering and random clustering of constraints.

Study B: Cognitive Complexities. There are different possibilities to represent equivalent semantics on the basis of a constraint, for example, the *requires* relationship $X \rightarrow Y$ can be represented in terms of $\neg X \vee Y$. The *incompatibility* relationship $\neg(X \wedge Y)$ can be represented as $X \rightarrow \neg Y$. Table 6 depicts five different possibilities to express *requires* and *incompatibility* relationships.

[1] We used these tasks to measure knowledge understanding. Further more differentiated tasks are within the scope of future work.

$Requires$	$Incompatibility$
$X \rightarrow Y$	$X \rightarrow \neg Y$
$\neg X \vee Y$	$\neg X \vee \neg Y$
$\neg Y \rightarrow \neg X$	$Y \rightarrow \neg X$
$\neg(X \wedge \neg Y)$	$\neg(X \wedge Y)$
$Y \leftarrow X$	$\neg Y \leftarrow X$

Table 6: Five different possibilities of representing *requires* and *incompatibility* relationships.

Study B is based on an within-subjects design (N=66) with two configuration knowledge bases. Knowledge base kbb_1 consisted of a set of *requires* constraints and kbb_2 consisted of a set of *incompatibility* constraints. Each study participant (again, computer science students who visited a related knowledge engineering course) had the task of finding a solution for the given CSP. Each participant was confronted with one version of kbb_1 and one version of kbb_2 conform the schema depicted in Table 6. For example, if a student received the $X \rightarrow Y$ version of kbb_1 then she/he also received the $X \rightarrow \neg Y$ version of kbb_2. The knowledge bases kbb_1 and kbb_2 were (again) defined in a domain-independent fashion (see Study A). The basic properties of the used knowledge bases are summarized in Table 7.

Knowledge base	#($v_i \in V$)	v_i domain size	#($c_i \in C$)
kbb_1	5	5	7
kbb_2	3	3	5

Table 7: Knowledge bases used in *Study B*.

The outcome of this experiment is shown in Table 8.

kbb_1*: SOL*	errors	kbb_2*: SOL*	errors
$X \rightarrow Y$	21.43%	$X \rightarrow \neg Y$	14.29%
$\neg X \vee Y$	50.0%	$\neg X \vee \neg Y$	34.62%
$\neg Y \rightarrow \neg X$	96.43%	$Y \rightarrow \neg X$	50.0%
$\neg(X \wedge \neg Y)$	73.08%	$\neg(X \wedge Y)$	42.31%
$Y \leftarrow X$	25.0%	$\neg Y \leftarrow X$	16.67%

Table 8: Error rates in solution identification (SOL) depending on constraint representation.

A result of the study is that basic implications ($\rightarrow$) should be preferred to other representations in order to maximize understandability. The only type of knowledge representation with a similar performance is the reverse implication, however, when comparing both alternatives, the standard implication seems to be the better choice.

4 Related Work

There is a long history of research on the improvement of knowledge engineering processes. Early research focused on model-based knowledge representations that allowed a separation of domain and problem solving knowledge. An example of such a representation are constraint technologies which became extremely popular as a technological basis for industrial applications [Freuder, 1997]. In a next step, graphical

knowledge representations [Felfernig *et al.*, 2000] and intelligent techniques for knowledge base testing and debugging have been developed [Felfernig *et al.*, 2004]. The need of an intuitive access to a corpus of software artifacts is also one of the major requirements for *software comprehension* [Storey, 2006]. In this context, recommender systems [Jannach *et al.*, 2010] have already been identified as a valuable means to provide intelligent support for the navigation in large and complex software spaces (see, e.g., [Robillard *et al.*, 2010]). The application of recommendation technologies for supporting knowledge engineering processes is a new research area. Research contributions in this field have the potential to significantly improve the overall quality of knowledge engineering processes. In [Felfernig *et al.*, 2010] basic knowledge representations are compared, for example, the use of $\rightarrow$ to represent an implication vs. the use of $\neg$ and $\vee$. This work is an important step towards a discipline of *empirical knowledge engineering* with a clear focus on usability aspects and cognitive efforts needed to complete knowledge engineering tasks. The work presented in this paper is a continuation of the work of [Felfernig *et al.*, 2010]. It takes a more detailed look at different alternative representations of *requires* and *incompatibility* relationships and introduces a new concepts related to the content-based clustering of constraints.

5 Conclusions

In this paper we showed how recommenders can be exploited to support knowledge engineering tasks. Examples are collaborative filtering of constraint sets, clustering of constraints, and knowledge-based recommendation of refactoring operations. Future work will include the development of further recommendation algorithms, for example, the inclusion of content-based filtering and further clustering algorithms as well as further empirical studies with more differentiated maintenance tasks. Finally, we will focus on an in-depth analysis of existing research in the area of cognition psychology which can further advance the state of the art in (configuration) knowledge engineering.

References

[Burke, 2000] R. Burke. Knowledge-based recommender systems. *Library and Inf. Systems*, 69(32):180–200, 2000.

[Felfernig and Burke, 2008] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *ACM International Conference on Electronic Commerce (ICEC08)*, pages 17–26, 2008.

[Felfernig *et al.*, 2000] A. Felfernig, G. E. Friedrich, and D. Jannach. UML as Domain Specific Language for the Construction of Knowledge-based Configuration Systems. *IJSEKE*, 10(4):449–469, 2000.

[Felfernig *et al.*, 2001] A. Felfernig, G. Friedrich, and D. Jannach. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering*, 15(2):165–176, 2001.

[Felfernig *et al.*, 2004] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.

[Felfernig *et al.*, 2009] A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible repairs for inconsistent requirements. In *21st Intl. Joint Conference on Artificical Intelligence (IJCAI'09)*, pages 791–796, Pasadena, CA, 2009.

[Felfernig *et al.*, 2010] A. Felfernig, M. Mandl, A. Pum, and M. Schubert. Empirical knowledge engineering: Cognitive aspects in the development of constraint-based recommenders. In *IEA/AIE 2010*, pages 631–640, Cordoba, Spain, 2010.

[Felfernig *et al.*, 2013] A. Felfernig, M. Schubert, and S. Reiterer. Personalized Diagnosis for Over-Constrained Problems. In *23rd International Conference on Artificial Intelligence*, Peking, China, 2013.

[Freuder, 1997] E. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997.

[Huffman and Kahn, 1998] C. Huffman and B. Kahn. Variety for Sale: Mass Customization or Mass Confusion. *Journal of Retailing*, 74:491–513, 1998.

[Jannach *et al.*, 2010] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems*. CUP, 2010.

[Junker, 2004] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *19th National Conference on Artificial Intelligence (AAAI04)*, pages 167–172, San Jose, CA, 2004.

[Konstan *et al.*, 1997] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[McDermott, 1982] J. McDermott. R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence Journal*, 19:39–88, 1982.

[Pazzani and Billsus, 1997] M. Pazzani and D. Billsus. Learning and revising user profiles: the identification of interesting websites. *Mach. Learn.*, 27:313–331, 1997.

[Robillard *et al.*, 2010] M. Robillard, R. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[Soloway, 1987] E. et al. Soloway. Assessing the Maintainabiliy of XCON-in-RIME: Coping with the Problem of very large Rule-bases. In *Proc. of AAAI-87*, pages 824–829, Seattle, Washington, USA, July 13–17 1987.

[Storey, 2006] M. Storey. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, 14:187–208, 2006.

[Stumptner *et al.*, 1998] M. Stumptner, G. Friedrich, and A. Haselböck. Generative Constraint-based Configuration of Large Technical Systems. *AI EDAM*, 12(4):307–320, 1998.

[Witten and Frank, 2005] I. Witten and E. Frank. *Data Mining*. Morgan Kaufman, 2005.

Solving Object-oriented Configuration Scenarios with ASP *

Gottfried Schenner and **Andreas Falkner**
Siemens AG Österreich, Vienna, Austria
gottfried.schenner@siemens.com
andreas.a.falkner@siemens.com

Anna Ryabokon and **Gerhard Friedrich**
Universität Klagenfurt, Austria
anna.ryabokon@aau.at
gerhard.friedrich@aau.at

Abstract

The main configuration scenarios occurring in the domain of technical products and systems are consistency checking, completing a partial configuration, reconfiguration of an inconsistent configuration and finding the best knowledge base for future reconfigurations. This paper presents OOASP - a framework for the description of object-oriented product configurators using answer set programming and shows that it is able to solve the different (re)configuration scenarios occurring in practice. Thus, it is a step forward to close the conceptual gap between logic-based and object-oriented approaches for product configuration.

1 Introduction

A configurator is a software system that enables the user to design complex technical systems or services based on a predefined set of components. In modern configuration systems, domain knowledge - comprising configuration requirements (product variability) and customer requirements - is expressed in terms of component types and relations between them. Each type is characterized by a set of attributes which specify the functional and technical properties of real-world and abstract components of the configurable product. An attribute takes values from within a predefined domain. Furthermore, components are related/connected to each other in various ways. Each component type has a number of ports which allow to connect a component of that type with other components. A possible connection between two component types is modeled as a relation and its cardinality expresses the number of components that can be connected to a port. In most cases, modeling languages used in configuration allow to specify relations of the following types: classification (is-a), composition (part-of), association (user defined relations).

For simple customer products, a configuration system aims at finding a consistent and complete configuration for a given set of customer requirements and reconfiguration is seldom an issue. Reconfiguration occurs during the maintenance of technical systems with a long life-span, where parts of existing configurations have to be adapted continuously.

Reconfiguration is especially challenging if an existing system has to be extended with new functionality that was not part of the original system design. In this case, some relations between new and existing components have to be created or some of the existing relations have to be changed in order to meet modified configuration requirements. [Falkner and Haselböck, 2013] discuss typical problems occurring when configuration requirements are changed. Finding the best design for future configurations is an important task for the reconfiguration scenario since it allows to reduce costs during the production process.

In the current paper we present a generic configurator which uses an object-oriented approach to encode its knowledge base. In order to compute configurations the system uses answer set programming (ASP). We illustrate the mapping from an object-oriented formalism (UML) to logical descriptions using a simplified real-world example from Siemens. Additionally, the paper provides different insights on (re)configuration scenarios such as checking and completing a configuration, reconciliation and choosing the best knowledge base for reconciliation. Finally, we discuss challenges which frequently occur in practice and should be taken into account while solving (re)configuration problems.

The remainder of this paper is organized as follows: In Section 2 we introduce a sample configuration problem used as example throughout the paper. After an ASP overview in Section 3, we describe in Section 4 how object-oriented knowledge bases can be specified using ASP. In Section 5 various product configuration scenarios are discussed. Section 6 provides some evaluation details and in Section 7 we conclude.

2 Configuration example

Modules example is a simple hardware configuration problem. Figure 1 shows the configurable objects of the example domain in a UML diagram: hardware frames contain up to five modules of various types (A, B) and elements of various types are assigned to the modules (one by one). Additionally to the cardinality constraints implied by the UML diagram there are the following domain-specific constraints:

- Elements of type ElementA require a module of type ModuleA

*This work has been developed within the scope of the project RECONCILE (reconciling legacy instances with changed ontologies) and was funded by FFG FIT-IT (grant number 825071).

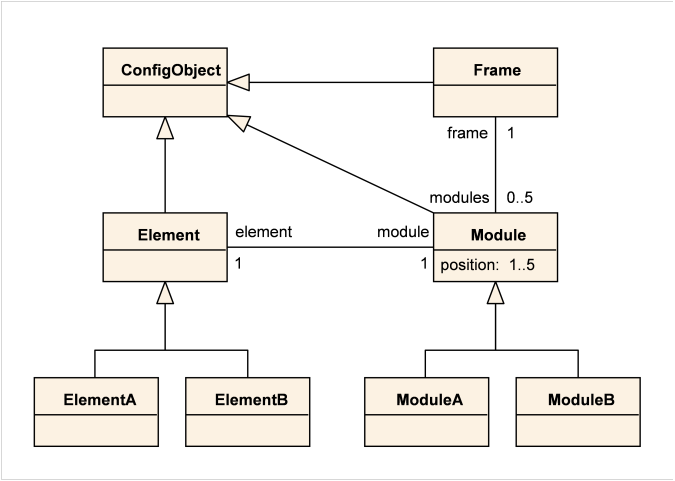


Figure 1: UML diagram for the modules example

- Elements of type ElementB require a module of type ModuleB
- The position of the modules of a frame must be unique

In a typical configurator scenario for this domain, a user creates a partial configuration consisting of elements of different types. Then the configurator extends the configuration by creating missing modules for the elements and by creating missing frames and assigning the modules to them. If the user manipulates a completed configuration, for instance by adding or removing elements, the configurator can restore consistency through reconfiguration, usually by keeping as much of the existing structure of the configured system as possible.

3 Answer set programming overview

Answer set programming is an approach to declarative problem solving which has its roots in logic programming and deductive databases. It is a decidable fragment of first-order logic interpreted under stable model semantics and extended with default negation, aggregation, and optimization. ASP allows modeling of a variety of (combinatorial) search and optimization problems in a declarative way using model-based problem specification methodology (see e.g. [Gelfond and Lifschitz, 1988; Eiter *et al.*, 2009; Brewka *et al.*, 2011] for details). ASP has a long history of being used for product configuration [Soininen and Niemel, 1998].

An ASP program is a finite set of rules of the form:

$$\mathtt{a} :- \mathtt{b}_1, \ldots, \mathtt{b}_m, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_k. \quad (1)$$

where $\mathtt{a}$, $\mathtt{b}_i$, and $\mathtt{c}_j$ are *atoms* of the form $\mathtt{predicate}(\mathtt{term}_1, \ldots, \mathtt{term}_n)$. A *term* is either a variable or a constant.

In most of ASP languages, variables are denoted by strings starting with uppercase letters and constants as well as predicates by strings starting with lower case letters. An atom together with its negation is called *literal*, e.g. `a` is a positive and `not a` is a negative literal. In the rule (1), the literal `a` is the *head* of the rule and the conjunction $\mathtt{b}_1, \ldots, \mathtt{b}_m, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_k$ is the *body*. A rule with an empty head, standing for *false*, is called an *integrity constraint*, i.e. every interpretation that satisfies the body of the constraint is not an answer set (configuration solution). A rule with an empty body is called a *fact*. Rule (1) derives that the atom `a` in the head of the rule is *true* if all literals of the body are *true*, i.e. there is a derivation for each positive literal $\mathtt{b}_1, \ldots, \mathtt{b}_m$ whereas none of the atoms of the negative literals $\mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_k$ can be derived.

Processing of a general ASP program P, in which atoms can contain variables, is done in two stages [Brewka *et al.*, 2011]. First the program is grounded, i.e. P is replaced by a possibly small equivalent propositional program $grnd(P)$ in which all atoms are variable-free. In the second stage an ASP solver is used to identify answer sets. Following the definition of configuration problems based on logical descriptions, presented in [Soininen *et al.*, 2001; Felfernig *et al.*, 2004], each configuration is a subset of a finite Herbrand-model. Given the stable model semantics used in ASP, a Herbrand interpretation I is a *model* of a program P *iff* (a) it satisfies all the rules in P, (b) for every atom $a_i \in I$ there exists a justification based on given facts and (c) I is minimal under set inclusion among all (consistent) interpretations.

In this paper we use an ASP dialect implemented in Gringo [Gebser *et al.*, 2011][1] which includes a number of extensions simplifying the presentation of the programs. Thus, it allows definition of *weight* constraints which are defined as $\mathtt{l}[\mathtt{a}_1 = \mathtt{w}_1, \ldots, \mathtt{a}_n = \mathtt{w}_n]\mathtt{u}$ where $\mathtt{a}_i$ are atoms, $\mathtt{w}_i$ are weights of the atoms and $\mathtt{l}, \mathtt{u}$ are integers specifying lower and upper bounds. Such constraints allow declaration of choices, i.e. such number of atoms from the set $\{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ must be true that the sum of corresponding weights is between `l` and `u`. If the lower or upper bounds are missing, then the ASP grounder substitutes $\mathtt{l} = 0$ and $\mathtt{u} = \mathtt{n}$, where `n` is the sum of the weights of all atoms in the set. A special case of the weight constraints are *cardinality* constraints where each weight $\mathtt{w}_i = 1$. Cardinality constraints are denoted by curly brackets.

ASP dialects include operators that are used for generating sets of atoms: The *range* operator ("..") is used to generate a set of atoms such that each atom includes one of the integer constants from a given range of integers. The *generate* operator (":") is used in weight constraints to create sets of atoms used in it.

Example Assume that we want to encode a simple problem instance of the modules example including two frames with ids 1 and 2, and six modules with ids ranging from 10 to 15. These customer requirements can be represented as facts:

```
frame(1..2). module(10..15).
```

The relation between modules and frames, i.e. that each module must be placed in exactly one frame, is encoded using a choice rule:

```
1{mod2fr(X,Y) : frame(Y)}1 :- module(X).
```

[1] Potassco ASP suite: http://potassco.sourceforge.net

When the rule above is grounded, the grounder generates six rules - one for each module. E.g., for module 10 it outputs:

```
1{mod2fr(10,1), mod2fr(10,2)}1 :- module(10).
```

In order to allow at most five modules to be put in a frame we add the following cardinality constraint to our program:

```
:- frame(X), 6 {mod2fr(Y,X) : module(Y)}.
```

Due to the cardinality constraint every configuration (answer set) containing a frame with more than 5 modules will be eliminated.

Identification of the preferred configuration solution can be done using the built-in optimization functionality of ASP solvers. In the ASP dialect used in this paper, the optimization is defined on a weighted set of true atoms and indicated via `#minimize` or `#maximize` statements.

4 OOASP framework

OOASP[2] is a framework for describing object-oriented knowledge bases in ASP. A knowledge base consists of the object model of the configurator and additional constraints which a valid configuration must satisfy. It is assumed that the object model of the object-oriented configurator can be described by an UML class diagram [Rumbaugh *et al.*, 2005]. The structure of a knowledge base and configurations are described by special ASP facts. This fact-based language can be seen as a domain specific language (DSL) for defining object-oriented knowledge bases and configurations on top of ASP. The DSL can represent multiple configurator knowledge bases and solutions in one ASP program. The OOASP framework provides a default implementation for the DSL, e.g. the interpretation of the fact-based language, in several program packages (*.lp files). If advanced features (such as multiple inheritance, automatic symmetry breaking) are required, the default implementation must be replaced with alternative implementations, whereas the OOASP-DSL can stay the same. The OOASP-DSL is largely independent of special ASP features and can therefore be easily translated to other formalisms (OWL/RDF, UML, etc.).

4.1 Defining the knowledge base

The knowledge base comprises an object-model describing types of available components and possible relations between them. In addition, it can include a number of constraints on types and relations. To define the object-model of the configurator with the OOASP-DSL, the following predicates are used where all IDs are considered to be unique within a knowledge base.

```
ooasp_class(KBID,CID)
```

- Defines a class in the knowledge base KBID [3].
  KBID is an id for a knowledge base and CID is an id for a class within the given knowledge base.

```
ooasp_subclass(KBID,CID,SUPERCID)
```

- Defines an inheritance hierarchy of classes. Although other interpretations are possible, in this paper the inheritance hierarchy is assumed to be a tree (single inheritance).

```
ooasp_assoc(KBID,ASSOCID,
            CID1,C1MIN,C1MAX,CID2,C2MIN,C2MAX)
```

- Defines the association between classes CID1 and CID2 within given cardinalities, i.e. for every instance of CID1 there exist at least C2MIN and at most C2MAX instances of CID2 in the association and vice versa.

```
ooasp_attribute(KBID,CID,ATTRID,
                {"string","integer","boolean"})
```

- Defines an attribute for the class CID with the given type.

```
ooasp_attribute_minInclusive(KBID,CID,
                             ATTRID,MINVALUE)
```

- Defines an optional minimum value for integer attributes

```
ooasp_attribute_maxInclusive(KBID,CID,
                             ATTRID,MAXVALUE)
```

- Defines an optional maximum value for integer attributes

```
ooasp_attribute_enum(KBID,CID,
                     ATTRID,ENUMVALUE)
```

- Defines an enum-value (a possible value) for a string attribute.

The mentioned predicates are sufficient to describe the object-model of a simple object-oriented configurator. Many features which can be additionally found in object-oriented systems such as initial values, constants, multi-valued attributes, ordered associations, etc. are currently missing in the framework, but these features are not relevant to the demonstration of the configuration scenarios presented in the paper. In practice, especially ordered associations and initial values are a convenient feature of object oriented product configurators.

Example The OOASP-DSL representation for the modules example corresponds to the following set of facts:

```
% modules example kb "v1"
% classes
ooasp_class("v1","ConfigObject").
ooasp_class("v1","Frame").
ooasp_class("v1","Module").
ooasp_class("v1","ModuleA").
ooasp_class("v1","ModuleB").
ooasp_class("v1","Element").
ooasp_class("v1","ElementA").
ooasp_class("v1","ElementB").

% class inheritance
ooasp_subclass("v1","Frame","ConfigObject").
ooasp_subclass("v1","Module","ConfigObject").
ooasp_subclass("v1","Element","ConfigObject").
ooasp_subclass("v1","ElementA","Element").
ooasp_subclass("v1","ElementB","Element").
```

[2] The ASP code for OOASP is available upon request from the first author

[3] To allow uppercase names, OOASP identifiers are strings, not constants

```
ooasp_subclass("v1","ModuleA","Module").
ooasp_subclass("v1","ModuleB","Module").

% attributes and associations
% class Frame
ooasp_assoc("v1","Frame_modules",
  "Frame",1,1,
  "Module",0,5).

% class Module
ooasp_attribute("v1","Module","position","integer").
ooasp_attribute_minInclusive("v1",
  "Module","position",1).
ooasp_attribute_maxInclusive("v1",
  "Module","position",5).

% class Element
ooasp_assoc("v1","Element_module",
  "Element",1,1,
  "Module",1,1).
```

4.2 Defining a configuration

A (partial) configuration is an instantiation of the object-model. A valid configuration is a configuration where no constraint is violated. It represents a buildable artifact of the configured system.

As with knowledge-bases, OOASP allows the representations of multiple configurations within one ASP program. We use the following predicates to define a (partial) configuration:

`ooasp_configuration(KBID, CONFIGID)`

- Declares that the configuration CONFIGID belongs to the knowledge base KBID. Every configuration has a unique ID and belongs to exactly one knowledge base.

`ooasp_isa(CONFIGID, CID, OBJECTID)`

- The object with the id OBJECTID is an instance of the class CID in the configuration CONFIGID. If an object is an instance of a class, it must also be an instance of one of its leaf classes (i.e. class without subclasses).

```
ooasp_associated(CONFIGID,ASSOCID,
                 OBJECTID1,OBJECTID2)
```

- The objects with the OBJECTID1 and OBJECTID2 are associated in the association ASSOCID in the configuration CONFIGID.

```
ooasp_attribute_value(CONFIGID,ATTRID,
                      OBJECTID,VALUE)
```

- The attribute ATTRID of the object OBJECTID has the value VALUE in the configuration CONFIGID.

Example The following configuration consisting of one frame, one module, and one element is not valid. It would be valid if the module 10 was a ModuleB.

```
ooasp_configuration("v1","c1").
ooasp_isa("c1","Frame",1).
ooasp_isa("c1","ModuleA",10).
ooasp_attribute_value("c1","position",10,5).
ooasp_isa("c1","ElementB",20).
ooasp_associated("c1","Frame_modules",1,10).
ooasp_associated("c1","Element_module",20,10).
```

4.3 Defining constraints

There are two different kinds of constraints in OOASP: integrity constraints and domain-specific constraints. Both are implemented as ASP rules which derive an atom `ooasp_cv` (head) for each constraint violation expressed in the body. The derived atom can be used for explanations.

Integrity constraints are generic constraints derived from the object model of the knowledge base. Implementations of integrity constraints are provided by the OOASP framework in program package ooasp_check.lp, e.g. for the constraint which checks the minimal cardinality of associations:

```
% Derive ooasp_cv(CONF,mincardviolated(ID1,ASSOC))
% whenever there are less objects associated
% with object ID1 than allowed by the cardinality
% restriction of the association
ooasp_cv(CONF,mincardviolated(ID1,ASSOC)) :-
  { ooasp_associated(CONF,ASSOC,ID1,ID2):
    ooasp_isa(CONF,C2,ID2) } C2MIN-1,
  C2MIN>0,
  ooasp_isa(CONF,C1,ID1),
  ooasp_assoc(KBID,ASSOC,
              C1,C1MIN,C1MAX,C2,C2MIN,C2MAX),
  ooasp_configuration(KBID,CONF).
```

In addition to the integrity constraints, a knowledge engineer can define domain-specific constraints for a knowledge base. These are constraints that can not be derived automatically from the knowledge base.

Example The first of the constraints in the modules examples may be implemented as follows:

```
% ElementA requires ModuleA
ooasp_cv(CONF,wrongModuleType(E,M)) :-
  ooasp_configuration("v1",CONF),
  ooasp_associated(CONF,"Element_module",E,M),
  ooasp_isa(CONF,"ElementA",E),
  not ooasp_isa(CONF,"ModuleA",M).
```

5 Product Configuration Scenarios

This section describes some of the typical scenarios for an object-oriented product configurator.

5.1 Checking a Configuration

Checking a (partial) configuration evaluates the integrity constraints of the knowledge base and the domain-specific constraints for a configuration under closed world assumption, i.e. during the checking no new objects are instantiated.

In an interactive configurator, checking the current configuration highlights the parts of the configuration that need to be changed by a user.

Example Checking the minimal configuration consisting of only one element of type A

`ooasp_isa("c2","ElementA",10).`

will derive a cardinality violation

`ooasp_cv("c2",mincardviolated(10,"Element_module")).`

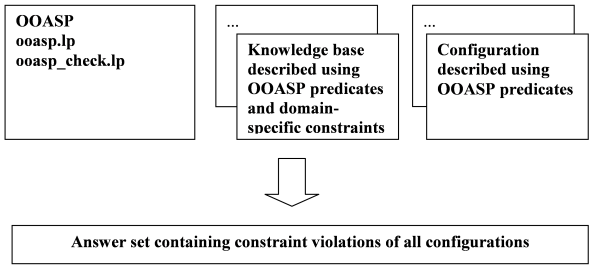


Figure 2: Checking a configuration

for the association between element and module, indicating that there must be a module for the element with OBJECTID 10.

The constraint violations derived during checking can also guide a repair-based solver to repair the current configuration [Falkner *et al.*, 2011]. If checking does not find any constraint violation, the current configuration is valid. The process of checking a configuration within OOASP framework is depicted in Figure 2.

5.2 Completing a Configuration

Given a possible empty (partial) configuration, find an extension of the configuration that satisfies all constraints. No fact of the given configuration may be removed. If no such extension can be found, the current configuration is either inconsistent or there is no valid configuration with the given upper bounds for object instances.

Completing a configuration can be accomplished by enumerating all possible extensions of the given configuration until a valid configuration is found. Figure 3 shows the necessary program packages for completing a configuration.

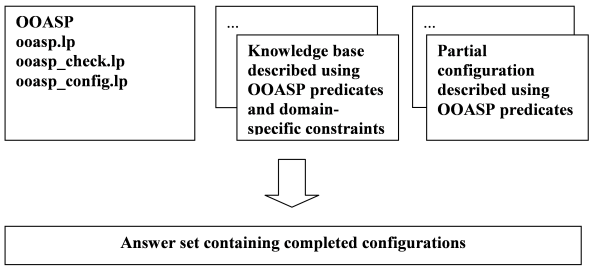


Figure 3: Completing a configuration

To enumerate all possible configurations one has to instantiate objects according to customer requirements. The number of the possible instances is controlled by the predicate

`ooasp_domain(CONFIGID, CID, OBJID)`

- The object with the OBJID can be instantiated to one of the leaf-classes of class CID.

The `ooasp_domain` facts define the available object IDs for a configuration. The object IDs are unique within a configuration. Every object ID can represent one instance of a leaf-class. However, the classes used in the `ooasp_domain` predicates can be non-leaf-classes as well. Therefore, the number of `ooasp_domain` facts for each class CID is equal to the maximal number of its instances in the configuration CONFIGID. From the `ooasp_domain` facts the possible types of every object ID in a configuration are derived (`ooasp_canbe`), searching up and down the class hierarchy (see (1) in Figure 4). The `ooasp_isa` facts (2) are derived upwards only. This approach of controlling instantiation is similar to the notion of a scope in Alloy [Jackson, 2011].

Example `ooasp_domain("c3","Module",20)` allows the object with OBJECTID 20 to become either a ModuleA or a ModuleB but not Frame. In a second step it may be set explicitly to be a ModuleA. Figure 4 shows the derived information after the object has been instantiated this way.

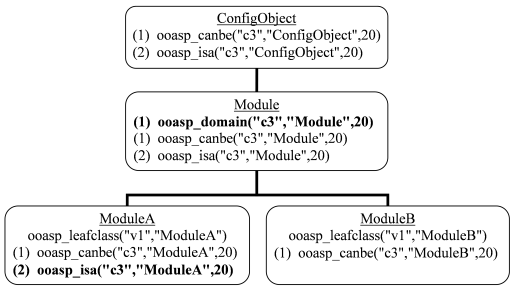


Figure 4: Controlling instantiation

The enumeration of all possible configurations is accomplished by instantiating objects and setting associations and attributes. The default implementation for instantiating objects is done in program package ooasp_config.lp:

```
% instantiate objects
0 { ooasp_isa(CONF,LEAFCLASS,ID) :
    ooasp_leafclass(V,LEAFCLASS) :
    ooasp_canbe(CONF,LEAFCLASS,ID) } 1 :-
  ooasp_domain(CONF,C,ID),
  ooasp_configuration(V,CONF).
```

This means that every object ID can become an instance of one of its possible leaf-class types. Associations are set in a similar matter. Every instance in the configuration can be associated with all other possible instances in the configuration. Constraints ensure that only instantiated objects are associated.

```
% associate objects
C2MIN { ooasp_associated(CONF,ASSOC,ID1,ID2):
        ooasp_canbe(CONF,C2,ID2) } :-
  ooasp_isa(CONF,C1,ID1),
  ooasp_assoc(V,ASSOC,C1,C1MIN,C1MAX,C2,C2MIN,C2MAX),
  ooasp_configuration(V,CONF).
```

```
% type check - only use instantiated objects
:- ooasp_associated(CONFIG,ASSOC,ID1,ID2),
   not ooasp_isa(CONFIG,C2,ID2),
   ooasp_assoc(V,ASSOC,C1,C1MIN,C1MAX,C2,C2MIN,C2MAX),
   ooasp_configuration(V,CONFIG).
```

Finally, there must be a generating rule for all possible values of the attributes of an object. The following shows the generating rule for integer attributes.

```
% set attribute values for integer attributes
1 { ooasp_attribute_value(CONFIG,N,ID,VALUE):
    VALUE=MIN..MAX } 1 :-
  ooasp_attribute(V,C,N,T),
  ooasp_isa(CONFIG,C,ID),
  ooasp_attribute_minInclusive(V,C,N,MIN),
  ooasp_attribute_maxInclusive(V,C,N,MAX),
  ooasp_configuration(V,CONFIG).
```

Example Figure 5 shows a completed configuration for the partial configuration `c3` below. It contains three instances of ElementA and two instances of ElementB. Note that `ooasp_isa` is given as customer requirement only for those elements. For modules, only the `ooasp_domain` is given and only 5 of the available 10 IDs are used in the completed configuration.

```
% Partial configuration
ooasp_configuration("v1","c3").
ooasp_domain("c3","Frame",1).
ooasp_domain("c3","ElementA",10..12).
ooasp_isa("c3","ElementA",10..12).
ooasp_domain("c3","ElementB",13..14).
ooasp_isa("c3","ElementB",13..14).
ooasp_domain("c3","Module",20..29).
```

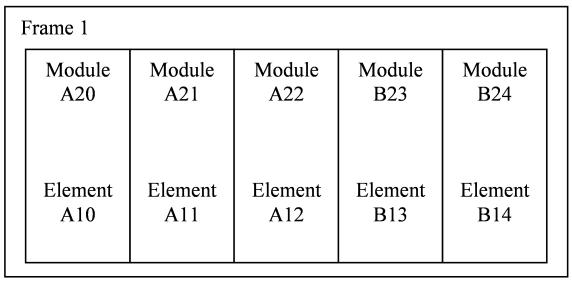


Figure 5: Complete configuration for the modules example

5.3 Reconciliation

Given a complete legacy configuration and the changed knowledge base which makes the configuration invalid, find a new valid configuration that is close to the legacy configuration.

Reconciliation of a configuration is illustrated in Figure 6. OOASP uses the same cost-based reconciliation approach as described in [Friedrich *et al.*, 2011]. For every change in the legacy configuration, a cost can be defined. This allows a fine control over the reconfiguration process. The optimal reconciliation is the reconfiguration that minimizes the costs. For example, the rule for reconciling associations either keeps the

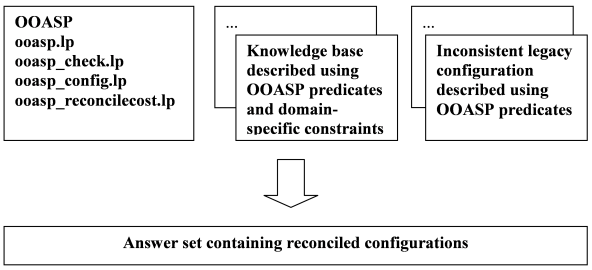


Figure 6: Reconcile a configuration

link between two objects in the legacy configuration or removes it. Reconciliation is controlled by the following predicates:

`ooasp_reconcile(LEGACY,RECONCILED)`

- Activates reconciliation from configuration LEGACY to the configuration RECONCILED

`ooasp_cost_instance(KB,CID,ADD,REMOVE)`

- Defines the costs for adding and removing instances of class CID

`ooasp_cost_assoc(KB,ASSOC,ADD,REMOVE)`

- Defines the costs for adding and removing a link to/from the association ASSOC

`ooasp_cost_attribute(KB,ATTR,COST)`

- Defines the cost for changing attribute ATTR

`ooasp_rcost(CHANGEOFLEGACYCONFIGURATION,COST)`

- For every modification of the legacy configuration an `ooasp_rcost` atom is derived, defining the COST of the modification. The best reconciliation is the one that minimizes the overall cost of the `ooasp_rcost` atoms, i.e. `#minimize[ooasp_rcost(CHANGE,COST)=COST]`.

The following listing shows the implementation of the rules for reconciling associations:

```
% either reuse link or remove it:
% ooasp_remove_associated is derived,
% if a link is removed
1 { ooasp_associated(RECONCILED,ASSOC,ID1,ID2),
    ooasp_remove_associated(RECONCILED,ASSOC,
                            ID1,ID2) } 1 :-
  ooasp_associated(LEGACY,ASSOC,ID1,ID2),
  ooasp_reconcile(LEGACY,RECONCILED).

% derive the reconfiguration costs
% ooasp_rcost contains the overall costs
ooasp_rcost(ooasp_remove_assoc(ID1,ID2),REMOVE) :-
  ooasp_remove_associated(RECONCILED,ASSOC,ID1,ID2),
  ooasp_cost_assoc(KB,ASSOC,ADD,REMOVE),
  ooasp_configuration(KB,RECONCILED),
  ooasp_reconcile(LEGACY,RECONCILED).
```

Example Suppose after the first systems of our example domain have been built, there is evidence of a previously unknown overheating problem if two modules of type A are put next to each other in a frame. Thus, to prevent overheating we have to add a new constraint to the knowledge base that disallows putting two modules of type A next to each other.

```
%  do not put 2 modules of type moduleA
%  next to each other
ooasp_cv(CONF,moduleANextToOther(M1,M2,P1,P2)):-
  ooasp_configuration("v2",CONF),
  ooasp_associated(CONF,"Frame_modules",F,M1),
  ooasp_associated(CONF,"Frame_modules",F,M2),
  ooasp_attribute_value(CONF,"position",M1,P1),
  ooasp_attribute_value(CONF,"position",M2,P2),
  M1!=M2,
  ooasp_isa(CONF,"ModuleA",M1),
  ooasp_isa(CONF,"ModuleA",M2),
  P2=P1+1.
```

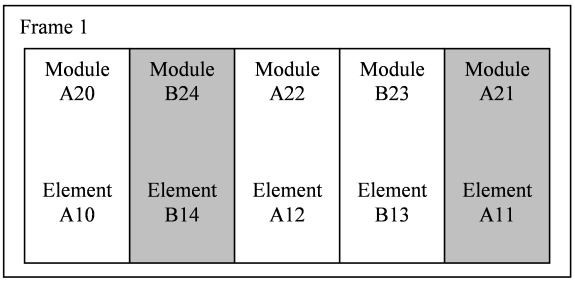


Figure 7: Reconciled configuration for the modules example

Because of the added constraint the legacy configuration in Figure 5 is no longer valid. Using reconciliation with equal costs for all changes to the configuration results in the new configuration shown in Figure 7.

5.4 Choosing the best knowledge base for reconciliation

Given a new technical requirement and N knowledge bases satisfying that requirement, choose the knowledge base that minimizes the costs for reconciling legacy configurations and the estimated costs for building a new system and maintaining existing systems.

Note that the costs for maintaining systems may also contain the costs for future reconciliations. Often there are many different technical solutions satisfying new requirements affecting existing systems. The choice of a technical solution that minimizes the costs for reconciliation of the legacy systems is an important problem to be solved. Given costs for various system modifications, we have to find a solution which corresponds to the most cost-effective reconciliation.

Example A possible technical solution for the overheating modules is to avoid putting the modules next to each other. Suppose there is an alternative technical solution replacing module A with a new module ANEW, which does not have the overheating problem.

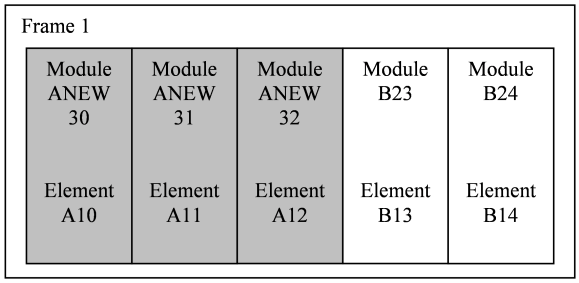


Figure 8: Reconciled configuration with the module of type ANEW

Reconcilation in the alternative knowledge base consists in replacing modules of type A with modules of type ANEW, but no rearranging is necessary. The result is shown in Figure 8. If modules of type A can be used together with type ANEW then it is sufficient to just replace module A 21 with module ANEW 31.

Which technical solution shall be chosen? To answer this question one has to find the affected configurations. With a framework like OOASP, the effected legacy configurations (i.e. deployed systems which must be reconfigured) can be computed by checking the constraint representing the new technical requirement in all available legacy configurations. Note that legacy configurations may use earlier versions of the knowledge base. In this case the legacy configurations must be upgraded to the current version of the knowledge base or the constraint must be expressed in terms of the legacy knowledge bases.

Using the reconcile scenario one can compute how costly it would be to modify the existing legacy configurations to the available technical solutions.

The cost for new systems can be estimated by computing the configuration cost of existing legacy systems, i.e. how costly it would have been to build these systems from scratch with the new knowledge bases. This can be computed by a configurator using the initial (partial) configuration, i.e. the customer requirements and completing the configuration with the new knowledge base.

The costs for future reconciliations are hard to compute in general, unless there is some knowledge about the future requirements. Otherwise, one has to estimate how often the critical constellations will occur. By concentrating on the most probable reconcile scenarios of a product configurator, one can simulate these reconciliation scenarios using alternative knowledge bases and compare their costs.

6 Evaluation

The main purpose of OOASP is to demonstrate the behavior of an object-oriented configurator within a logical framework. Therefore, performance was not the main focus of this paper. Since OOASP uses a similar approach as [Friedrich *et al.*, 2011], its performance is similar, too.

For checking configurations, the framework proved to be able to handle more than 1000 components for integrity con-

straints and simple domain-specific constraints. Of course, one can always come up with complex constraints, like finding all paths in a graph, for which computation of logical models is infeasible.

Completing a configuration can be handled for problems with hundreds of components. The main limiting factor here is the grounding size. The grounding explodes since the generic translation of UML to ASP rules of the default OOASP implementation associates every possible object of one related type with every other possible object of the other type. The grounding of the module example with 200 objects is already greater than 500 MB. One can reduce the grounding size by replacing the rules of the generic translation with special instantiation rules as follows:

```
% associate objects
% special implementation
% 'create' a module for every element
% at a fixed object ID
1 {ooasp_associated(CONF,
    "Element_module",
    ID1,1000+ID1)} 1:-
  ooasp_isa(CONF,"Element",ID1),
  ooasp_configuration(V,CONF).
```

This is similar to automatically generating subobjects in an object-oriented setting. However, these special rules can no longer be used for reconfiguration, because they assume that for every element there is a unique module at a fixed object ID. Another way to avoid the explosion of grounding size would be to use a constraint-based model. Additional limiting factor is the current lack of ASP to incorporate domain-specific heuristics into the solving, which is a topic of active research.

7 Conclusions

This paper demonstrates the implementation of a small object-oriented product configurator on top of ASP. The framework contains a domain-specific language for specifying knowledge bases and configurations, that can be easily translated to other formalisms (OWL/RDF, UML/Java).

Evaluations showed that checking constraints relative to a given configuration can be done effectively. However, finding (re)configurations efficiently remains a challenge for large-scale product configuration. The main obstacle for SAT- and ASP-based approaches seems to be the explosion of grounding. In addition, the identification of appropriate domain-specific heuristics is an open problem for all search-based approaches.

By defining typical configuration scenarios we hope to raise awareness to often neglected aspects of product configuration. We demonstrated the handling of these scenarios in ASP and are going to continue this work for other formalisms such as constraint programming, RDF/OWL, etc.

References

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[Eiter *et al.*, 2009] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, pages 40–110, 2009.

[Falkner and Haselböck, 2013] Andreas Falkner and Alois Haselböck. Challenges of Knowledge Evolution in Practice. *AI Communications*, 26:3–14, 2013.

[Falkner *et al.*, 2011] Andreas Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. Modeling and solving technical product configuration problems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25:115–129, 2011.

[Felfernig *et al.*, 2004] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.

[Friedrich *et al.*, 2011] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. (Re)configuration based on model generation. In Conrad Drescher, Ins Lynce, and Ralf Treinen, editors, *LoCoCo*, volume 65 of *EPTCS*, pages 26–35, 2011.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):105–124, 2011.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *5th International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.

[Jackson, 2011] D. Jackson. *Software Abstractions: Logic, Language and Analysis*. Mit Press, 2011.

[Rumbaugh *et al.*, 2005] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2 edition, 2005.

[Soininen and Niemel, 1998] Timo Soininen and Ilkka Niemel. Formalizing configuration knowledge using rules with choices. In *Seventh International Workshop On Nonmonotonic Reasoning*, 1998.

[Soininen *et al.*, 2001] Timo Soininen, Ilkka Niemelä, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *1st International Workshop on Answer Set Programming: Towards Efficient and Scalable Knowledge*, pages 195–201, 2001.

Configuring Domain Knowledge for Natural Language Understanding

Matt Selway and **Wolfgang Mayer** and **Markus Stumptner**
University of South Australia
Adelaide
{<first_name>.<last_name>}@unisa.edu.au

Abstract

Knowledge-based configuration has been used for numerous applications including natural language processing (NLP). By formalising property grammars as a configuration problem, it has been shown that configuration can provide a flexible, non-deterministic, method of parsing natural language. However, it focuses only on *syntactic* parsing. In contrast, configuration is usually performed using knowledge about a domain and is semantic in nature. Therefore, we argue that configuration has the potential to be used, not only for syntactic processing, but for the semantic processing of natural language, effectively supporting Natural Language Understanding (NLU).

In this paper, we propose an approach to NLP that applies configuration to the (partial) domain model evoked by the processing of a sentence. This has the benefit of ensuring the meaning of the sentence is consistent with the existing domain knowledge. Moreover, it allows the dynamic incorporation of domain knowledge in the configuration model as the text is processed. We demonstrate the approach on a business specification based on the Semantics of Business Vocabulary and Rules.

1 Introduction

Knowledge-based configuration has been used in numerous applications. While historically used for configuring physical products, configuration has been applied to other domains such as software services, software product lines, and constraint-based language parsing [Hotz and Wolter, 2013].

In particular, [Estratat and Henocque, 2004; Kleiner *et al.*, 2009] have applied configuration to a translation of property grammars (a constraint-based linguistic formalism). By formalising property grammars as a configuration problem, they show that configuration can provide a flexible, non-deterministic method for parsing natural language. However, these approaches focus on syntactic parsing by using the configuration process to generate a parse tree. In contrast, configuration is usually applied to domain knowledge, that is, semantic processing. Furthermore, in [Kleiner *et al.*, 2009] additional processes are required in order to transform the parse tree into a semantic model to make use of the domain knowledge. This causes some issues in ensuring the consistency and correctness of the domain knowledge.

In this paper we present an approach to parsing natural language that performs semantic processing directly using configuration. Instead of a model of language *categories* (e.g. noun, verb, noun phrase) and the *properties* (or constraints) on those categories, such as property grammars, we use a model of domain concepts and the relations between them. As a result, we perform natural language *understanding*; at least with respect to the semantic model being used.

Our approach maintains the advantages of using configuration for natural language processing, while gaining the following: (1) a simplified lexicon containing minimal lexical information, (2) improved consistency of the domain knowledge as the configuration process ensures its consistency during parsing, and (3) the semantic disambiguation of terms.

As our aim is to support the translation of informal natural language business specifications into formal models, we demonstrate our approach on an example from the business domain. The example business specification is defined using the Semantics of Business Vocabulary and Business Rules (SBVR) [OMG, 2008], which we use as our semantic model. SBVR and the example are discussed in more detail later.

The remainder of this paper is organised as follows: Section 1.1 provides a brief introduction to SBVR and its concepts, Section 2 presents an example that will be used throughout the paper, Section 3 describes our approach to parsing natural language, Section 4 presents experimental results, Section 5 discusses related work, and Section 6 provides insight into future work and concludes the paper.

1.1 Brief overview of SBVR

The Semantics of Business Vocabulary and Business Rules (SBVR) is a standard developed by the Object Management Group (OMG) to facilitate the transfer of business knowledge between business people and the technical experts responsible for developing information systems for them [OMG, 2008]. It encompasses two aspects: (1) a meta-model for representing vocabularies, facts, and rules in a machine processable format, and (2) a controlled English notation for representing the same vocabularies, facts, and rules more suited to people. Therefore, SBVR supports the exchange of business specifications between both organisations and software tools.

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

The SBVR meta-model standardises concepts for the definition of business vocabularies (i.e. sets of concepts relevant to a particular organisation or business domain) and rules relating to those vocabularies. It is based on formal logic; primarily first-order predicate logic with an extension in modal logic (necessity, possibility, permissibility, and obligation).

Within the meta-model, vocabularies are defined on the basis of Meanings and Representations. They consist of sets of interrelated *object types*, *individual concepts*, and *fact types*. A distinction is made between a meaning and its representation, allowing a single concept to be represented with multiple words (possibly in different languages), images, or sounds.

The semantic structure of business rules are formed by the Logical Formulations aspect of the meta-model. This includes concepts for first-order logical operators (e.g. conjunction, disjunction, implication), quantification (e.g. universal, existential, exactly n), and modal operators (e.g. necessity, obligation). These concepts allow business people to define structural and operative rules. Structural rules include such rules as cardinality constraints on the relations between concepts and cannot be violated, while operative rules may be violated by a person involved in conducting the business.

2 Motivating Example

This section introduces an example that identifies the limitations of existing approaches and that will be used in the remainder of this paper to demonstrate our approach. It is an extract of the EU-Rent business specification included in the SBVR specification [OMG, 2008, Annex E].

EU-Rent is a fictional car rental company with a global presence. The example business specification defines domain specific vocabulary and rules for EU-Rent, its structure, and how it conducts its business. Figure 1 shows a portion of the vocabulary related to the organisational structure of EU-Rent. The following is a structural rule, based on this vocabulary, that defines a cardinality constraint on the part-of relationship between a 'branch' and a 'local area'.

(1) *Each branch is included in exactly one local area.*

rental organisation unit
  Definition: organisational unit that operates part of EU-Rent's car rental business
rental organisation unit *having rental responsibility*
  Definition: the rental organisation unit is responsible for the operation of customer-facing rental business
rental organisation unit *having area responsibility*
  Definition: the rental organisation unit includes organisation units for which it has the responsibility to coordinate operations and ensure resources
local area
  Definition: rental organisation unit that *has area responsibility*
branch
  Definition: rental organisation unit that *has rental responsibility*
branch *is included in* local area
  Synonymous Form: local area *includes* branch

Figure 1: Business vocabulary used by the example rule with SBVR markup: object types, *fact types*, and keywords.

Although quite simple, this example demonstrates a number of important concepts such as nouns, verbs, and quantifiers. The approach of [Kleiner *et al.*, 2009] processes the sentence by executing a series of model transformations that result in a UML model of the text, using SBVR as an intermediate model between the natural language text and UML. The steps up to the creation of the SBVR model are as follows:

1. a text-to-model transformation creates an Ordered Words model that annotates the words with their position in the sentence
2. a model-to-model transformation creates a Labelled Words model by labelling each word with their possible syntactic categories using a lexicon (model)
3. *configuration* is used to transform the Labelled Words model into a Syntax model, performing syntactic and grammatical analyses, and
4. a model-to-model transformation creates an SBVR model from the Syntax model

Performing this process on the example sentence would result in the Syntax and SBVR models displayed in Figure 2.

This approach provides a flexible, non-deterministic, and extensible method of parsing natural language [Kleiner *et al.*, 2009]; however, it has several issues, chiefly: (1) it is primarily *syntactic*, (2) it requires a detailed lexicon, and (3) the mapping to SBVR can be problematic, e.g. in the handling of 'local area' two correct interpretations can be conceived.

The first is the main issue as, although [Estratat and Henocque, 2004] suggest that configuration can combine syntactic and semantic analysis, it is primarily used for generating syntactic parse trees. As a result, a sentence could be syntactically correct but not meaningful and, therefore, must be linked to the semantics somehow (e.g. through a model transformation to SBVR like that used in [Kleiner *et al.*, 2009]).

Although [Kleiner *et al.*, 2009] introduce some semantic elements into their model (i.e. each category can be linked to a basic element of the SBVR model[1]), they do so only to ease the transformation to the SBVR model. The existence of the SBVR elements does not provide any semantic guarantees. Therefore, semantic inconsistencies need to be resolved either during the transformation to SBVR, making it much more complex, or by post-processing of the SBVR model. However, this seems unnecessary if it can be achieved during the configuration process itself.

The requirement of a detailed lexicon is more an issue for our target application area than with the parsing method itself. In the context of businesses creating and maintaining their own sets of domain specific business vocabularies and business rules, we do not see business people defining detailed lexicons with linguistic information such as voice, genre, transitivity, etc. In this application area, the business vocabulary is more like a glossary containing domain specific words and their definitions, like that of Figure 1, rather than a dictionary containing detailed lexical information. Therefore, an approach that requires less linguistic information to be defined is required for our purposes.

[1] This is not shown in Figure 2a for readability.

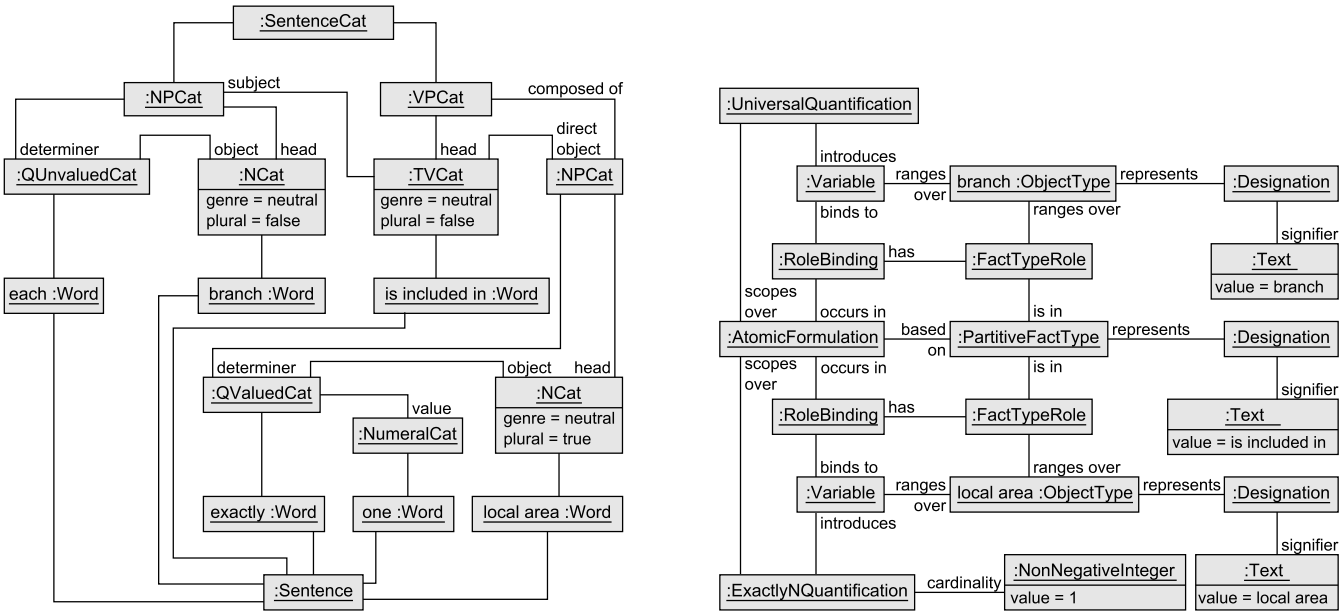


Figure 2: Example Syntax model (a) and SBVR model (b) generated by the process of [Kleiner *et al.*, 2009]

Finally, there are some problems with mapping the syntactic tree to the SBVR semantics. Consider how the term 'local area' is handled in the above example. For simplicity, 'local area' is a single noun, which maps directly to the object type 'local area' in SBVR. However, in reality it would be considered a noun phrase, where the term 'local' would be used in an adjective sense and 'area' would be the noun. This would map to the object type 'area' with the characteristic 'being local'. However, in the vocabulary of EU-Rent, 'local area' is a single concept and should not be decomposed in this way.

It seems a simple problem to fix: the term 'local area' could be included as a noun in the lexicon, which is what would happen if a business were defining its vocabulary. However, unless the individual terms 'local' and 'area' were removed, which could affect the processing of sentences in other contexts, it would result in two correct parses of the sentence: one in which 'local area' is treated as a simple noun and one treating it as a noun phrase. The transformation to the SBVR model would not resolve this issue either, as both forms have a valid mapping. Therefore, the user would have to select the preferred mapping or the SBVR model would have to be processed to see if either one or the other has already been created. Either way it makes the process more cumbersome, whereas we propose that by configuring the SBVR model directly, this problem would be avoided (as long as only one or the other has been specified in the vocabulary).

It could be argued that this problem is a quirk of the SBVR model as other semantic representations with a structure more similar to the parse tree would have more direct mappings. However, it is an important issue as SBVR has gained traction in our application domain in recent years [Nemuraite *et al.*, 2010; Sukys *et al.*, 2012] and is an important part of the OMG's Model-Driven Architecture [OMG, 2008]. Moreover, we will show that our proposed approach does not reduce the flexibility with respect to the semantic representation used.

3 Parsing Process

In order to overcome these limitations we propose the use of knowledge-based configuration on the semantic representation directly, rather than configuring a syntactical parse tree. In this way we maintain the benefits of parsing using configuration, while ensuring semantic consistency and removing a step from the process. Furthermore, this approach remains agnostic with respect to the semantic representation used; although we utilise the SBVR meta-model in this paper.

In order to avoid complex syntactical analysis, which is a difficult problem in itself, we use an approach inspired by Cognitive Grammar, a theory of grammar from the field of Cognitive Linguistics [Langacker, 2008]. Cognitive Grammar takes a holistic view of language, combining syntax, semantics, and pragmatics into a unified whole. In particular, our approach is based on that of [Holmqvist, 1993], which provides a computational model of Cognitive Grammar.

In Cognitive Grammar, the meaning of an expression is understood by combining the semantic structures evoked by its constituent expression into a unified structure; a process called *semantic accommodation* in [Holmqvist, 1993]. Evoking the semantic structure of an expression is a relatively simple endeavour as the two are linked; therefore, a semantic structure is evoked by looking up the expression in a lexicon. As a result, our method is able to do away with traditional syntactic analysis for a much simpler model, leaving most of the effort in understanding the expression to be performed by the accommodation process. With the aim of combining semantic structures into a composite structure based on the allowable relationships between them, the accommodation pro-

cess is analogous to a configuration task.

The syntactic analysis and the accommodation process using configuration are detailed in the following sections. These two processes are performed iteratively with the first using the expectations, or placeholders in our case, to propose possible parses of the sentence, and the second combining the semantic aspects of the suggested parses into a complete structure. The result is a progressively more detailed and complete set of domain knowledge containing the concepts, their definitions, and their associated rules.

3.1 Syntactic Analysis

The syntactic analysis of our approach is primarily the evocation of semantic structures from the lexicon, taking into account grammatical dependencies. For example, in the example expression, there are two quantifiers 'each' and 'exactly one'. If only the evoked semantic structures were known, the configurator would not know which quantifier applies to what concept, yet we know that 'each' is supposed to apply to the term 'branch' and 'exactly one' to 'local area'.

Traditionally, these dependencies are handled by the relationships between categories (as shown in Figure 2a). However, as this leads to complex lexicons that are not suitable for our target application and require complex processing to produce, we account for these dependencies using so called *grammatical expectations* [Holmqvist, 1993]. Grammatical expectations are a relatively simple model of grammatical dependency that identify locations in an expression where other expressions are "expected" to fill. Moreover, grammatical expectations are a good match for SBVR-based semantics, as they are similar to the placeholders of fact types. In [Holmqvist, 1993], only left and right expectations were introduced, which search to the left or right for another expression. We also introduced internal expectations, which search within the span of the expression, in order to more easily deal with SBVR fact types with more than two placeholders.

This model defines lexicon and lexical entries as follows.

Definition 1 (Lexicon)**.** A lexicon is a tuple
$l = (E, LE, lookup)$
where E is a set of expressions, LE is a set of lexical entries, and $lookup : E \rightarrow 2^{LE} \backslash \emptyset$

Definition 2 (Lexical Entry)**.** A lexical entry is a tuple
$le = (e, ss, GE, type_{GE})$
where e is an expression of one or more words, ss is its associated semantic structure, GE is a set of grammatical expectations, and $type_{GE} : GE \rightarrow \{left, internal, right\}$ assigns a type to each grammatical expectation $ge \in GE$.

This definition is purposefully generic with respect to the form the semantic structure takes. While we use the SBVR meta-model, other semantic representations could be used. As a result, our approach remains flexible in terms of varying the model being configured, as in [Kleiner *et al.*, 2009].

The lexicon is partly predefined. For example, words with explicit semantics in SBVR, such as those for quantifications, logical operators, etc., are explicitly defined as certain expressions, semantic structures, and grammatical expectations. Domain specific terms are provided by the vocabulary of a business specification, e.g. a glossary of terms, which have their semantic structures and grammatical expectations determined by the representation of object types and fact types in SBVR. In our application domain, the vocabulary is provided by business people, which drives our need for a simple lexicon with minimal information.

Using grammatical expectations, the syntactic analysis is performed incrementally, when an expression is found to fill an expectation (i.e. the expectation is said to *catch* the expression [Holmqvist, 1993]) a possible parse is proposed and kept in a suggestion list. Since the number of suggestions increases rapidly, the suggestion list is kept short by ordering the suggestions using heuristics to identify the *best* parse and pruning off any suggestions over a limit and/or that are not considered good candidates [Holmqvist, 1993]. The metrics used by the heuristics include: (1) catching distance, the linear distance to the word (or combination) filling a placeholder; (2) binding energy, the summation of all catching distances in a suggestion; (3) local coverage, the ratio of words in the suggestion to words spanned by the suggestion; and (4) global coverage, the ratio of words in the suggestion to all words of the expression encountered up to the current point.

The suggestion list and suggestions are defined as follows.

Definition 3 (Suggestion List)**.** A suggestion list SL is an ordered set of suggestions, where each suggestion $s \in SL$ is a tuple $s = (SLE, C, be, lc, gc)$ such that:

- SLE is a set of lexical entries or previous suggestions included in s
- C is a set of tuples
  $$catch = (sle_1 \in SLE, ge, sle_2 \in SLE, cd)$$
  that associate a grammatical expectation, ge, of the catching lexical entry or suggestion, sle_1, to the lexical entry or suggestion that it catches, sle_2, with the catching distance, cd.
- $be = \sum_{c \in C} c.cd$ is the binding energy of the suggestion,
- lc is the local coverage of the suggestion,
- gc is the global coverage of the suggestion

The order for each $s_1, s_2 \in SL$ is determined by:
$$s_1 \preceq s_2 \iff (s_1.gc, s_1.lc, s_1.be) \leq_l (s_2.gc, s_2.lc, s_2.be)$$

Based on the preferences that: the best parse should cover the entire sentence; suggestions should have no holes; and, words should be captured at the shortest distance.

Due to the recursive compositional nature of the suggestions, i.e. each $x \in SLE$ is a lexical entry or another suggestion, the catching information constitutes a non-traditional parse tree. Figure 3 shows an example of a parse tree produced by our syntactic analysis compared to the traditional parse tree equivalent to Figure 2a. The parse tree and the partial SBVR model (i.e. the semantic structures) of the suggested parse are provided to the semantic accommodation process to be configured into a complete model.

The general algorithm for the syntactic analysis is as follows, for each word in the expression:

1. Retrieve the entry for the current word from the lexicon
2. If the current word has any left placeholders, for each suggested parse in the suggestion list:

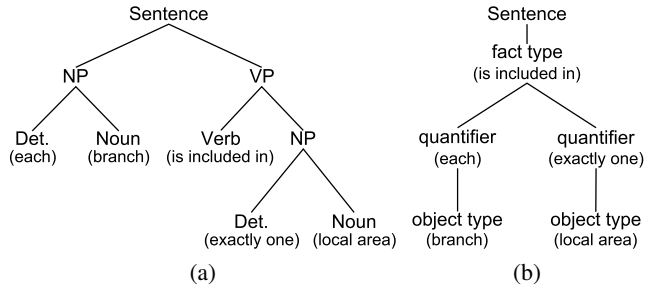


Figure 3: A traditional parse tree (a) and one created by our analysis (b)

   (a) catch the closest word (or word combination) to the left of the current word
   (b) add the new suggested parse to the suggestion list
3. Else, for each suggested parse in the suggestion list, if the previous word or combination has any internal placeholders and the current word is within its span:
   (a) catch the current word with the internal placeholder
   (b) add the newly suggested parse to the suggestion list
4. Else, for each suggested parse in the suggestion list, if the previous word or combination has any right placeholders:
   (a) catch the current word with the right placeholder
   (b) add the newly suggested parse to the suggestion list
5. Update the heuristics, distances between words, order the suggestion list, and cull excess entries
6. Provide newly suggested parses to the semantic accommodation/configuration process
7. Remove suggestions that failed accommodation

An example of the syntactic analysis after a complete parse of the example sentence is displayed in Figure 4.

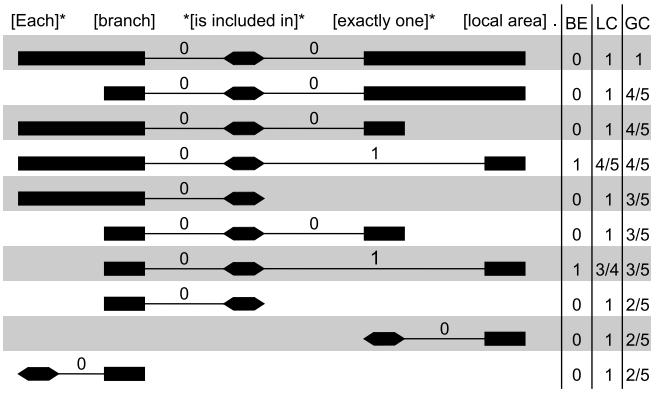


Figure 4: Lexical analysis of the rule 'Each branch is included in exactly one local area.' Asterisks represent the grammatical expectations, hexagons represent the catching word, rectangles represent the caught word, catching distance is shown above each line.

3.2 Semantic Accommodation/Configuration

Parses suggested by the syntactic analysis are sent to the semantic accommodation process to determine whether or not they are admissible in the (SBVR) semantics. Rather than the numerous processes for accommodation discussed in [Holmqvist, 1993], we utilise knowledge-based configuration to perform the accommodation. Specifically, component-oriented configuration is used, which combines advantages from connection-, resource-, and structure-based approaches [Soininen *et al.*, 1998; Stumptner, 1997]. Moreover, the object-oriented nature of component-oriented configuration lends itself more easily to that of the SBVR meta-model.

Using the terminology of [Soininen *et al.*, 1998], the SBVR meta-model constitutes the *configuration model knowledge* of our approach, i.e. it defines the types of entities, properties, and rules that specify the set of correct configurations. It follows that an SBVR (terminal) model constitutes the *configuration solution knowledge* or (possibly partial) *configuration*. Lastly, the parse tree created by the lexical analysis constitutes the *requirements knowledge*, i.e. additional constraints on the configuration that are not strictly part of the configuration model. For example, the SBVR meta-model may allow either quantification to be applied to either object type in the example rule, however, the grammatical dependencies require that 'each' be applied to 'branch' and 'exactly one' to 'local area' for the correct interpretation of the sentence.

Since the SBVR (meta-)model is defined using ECore, the Eclipse Modelling Framework [2] implementation of EMOF from the MOF specification of the OMG [OMG, 2006], we first discuss its mapping to the configuration ontology of [Soininen *et al.*, 1998]. The configuration ontology defines standard concepts for representing the different aspects of configuration knowledge including: taxonomy, attributes, structure, topology, and constraints. An example of the mapping for SBVR is displayed in Figure 5. It focuses on the configuration model knowledge, with some example component individuals. For simplicity, port individuals are not included in the figure. The mapping is by no means complete, but provides a link between the meta-model representation and the representation used for the configuration task.

Taxonomy

ECore allows classification hierarchies to be defined through the use of the concepts `EClass` and `EObject`, and the relations *eSuperTypes* and *eClass*. The concept `EClass` generically represents a type and therefore could be mapped to `Configuration Type`; however, ECore does not distinguish the sub-types `Component Type`, `Port Type`, `Resource Type`, and `Function Type` in the same manner as [Soininen *et al.*, 1998]. Therefore, it is more appropriate to map instances of `EClass` to `Component Type`s. Other ECore concepts map to the other configuration types. Therefore, only component types have a classification hierarchy; the others are made direct subtypes of their appropriate configuration type (i.e. `Attribute Type`, etc.).

Sub-types and super-types in ECore are represented by *eSuperTypes*. This relation defines the direct super-types of an

[2] `http://www.eclipse.org/modeling/emf/`

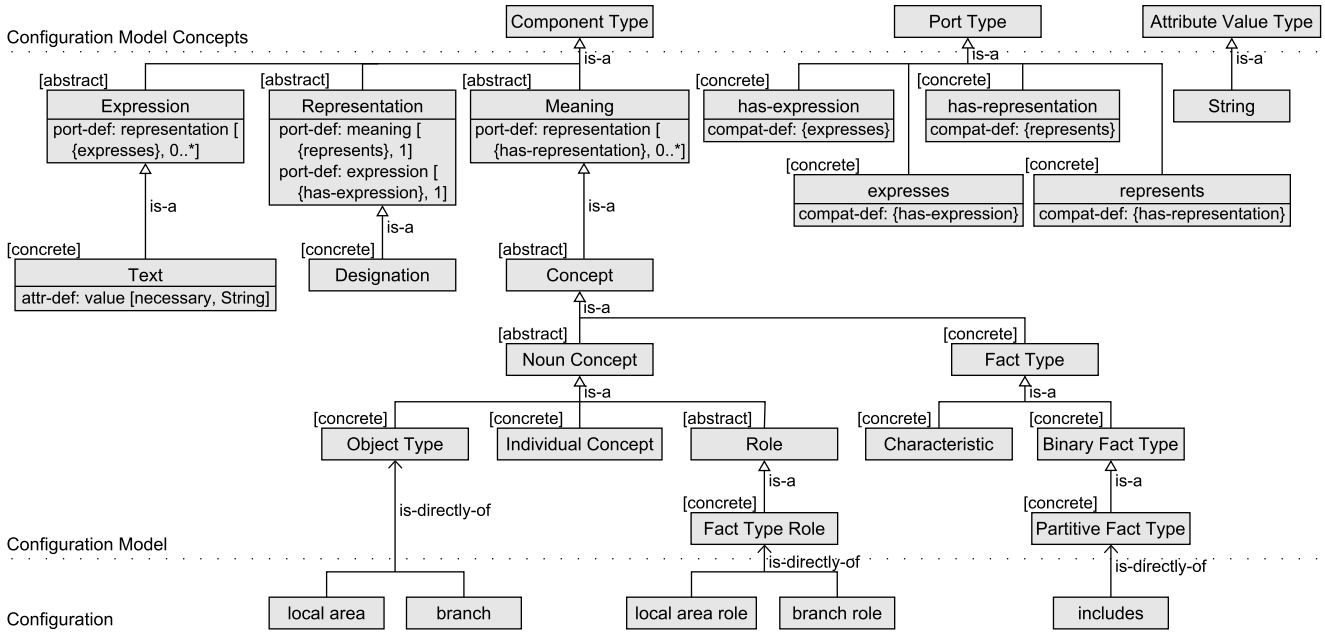


Figure 5: Fragment of the configuration model derived from the ECore mapping of the SBVR meta-model.

`EClass` and, therefore, maps to the *isa* relation. Moreover, multiple inheritance is allowed in both representations.

In ECore, an `EClass` may be abstract, i.e. cannot have any instances. However, in the context of configuration, it makes sense to relax this definition to allow partial information of a configuration, such as in [Soininen *et al.*, 1998]. Therefore, abstract and non-abstract (i.e. concrete) types in ECore are mapped to abstract and concrete classes in the ontology using the appropriate `Abstraction Definition`.

Instances of `EObject` represent `Individuals` from the ontology. In ECore, these are associated to their type by *eClass*, which maps to *is directly of*. Moreover, since *eClass* specifies the `EClass` of an individual, `EObject` necessarily maps to `Component Individual`.

Attributes

Attributes in ECore are represented by the concept `EAttribute`, which have a *name*, a type specified by *eAttributeType*, and relations for the lower and upper bounds of their cardinality (*lowerBound* and *upperBound*, respectively).

An *eAttributeType* links an `EAttribute` to its `EDataType`, which maps to the concept `Attribute Type` from [Soininen *et al.*, 1998]. It follows that `EAttributes` map to `Attribute Definitions` with the appropriate `Attribute Name`, `Attribute Value Type` (from *eAttributeType*), and `Necessity Definition`. The value of an attribute for a specific `EObject` is mapped to an `Attribute` with the respective `Attribute Value` and `Component Individual`.

In ECore, attributes can have a zero-to-many cardinality, while `Necessity Definitions` are restricted to exactly one (necessary) and at most one (optional) attributes. As a result, there exists only a partial mapping to the configuration ontology; however, this is not a problem for the SBVR meta-model as it only includes necessary and optional attributes.

Structure and Topology

The ontology of [Soininen *et al.*, 1998] differentiates between `Part Definitions`, which specify the compositional structure of components, and `Port Definitions`, which specify the topological connections (either physical or logical) between components. `Part Definitions` constitute a direct *has-part* relation between components. This relation must be anti-symmetric and anti-reflexive. Moreover, the transitive closure of *has-part* defines a *transitive has-part* relation, which must also be anti-symmetric and anti-reflexive. `Port Definitions` have no such restriction.

In ECore, both `Part Definitions` and `Port Definitions` are represented by the concept `EReference`. An `EReference` may be a *containment* reference (for compositional relationships) and/or it may have an *eOpposite* for bi-directional relationships.

While it seems intuitive to map containment and uni-directional references to part definitions, and bi-directional relationships to port definitions, this is not possible as ECore does not uphold the anti-symmetric and anti-reflexive requirements of *has-part* relations. For example, the situation shown in Figure 6, in which the transitive closure of the *has-part* relations between Meaning, Representation, and Expression are reflexive, is allowable in ECore but not the configuration ontology. To determine those `EReferences` that could be mapped to part definitions would require analysis of the meta-model; instead, we map all `EReferences` to ports. As a result, we effectively use ports as a generalised structural relationship similar to that described in [Hotz and Wolter, 2013].

A `Port Definition` requires a `Port Name`, a `Possible port type set`, and a `Cardinality`. Similar to `EAttribute`, `EReferences` have a *name*,

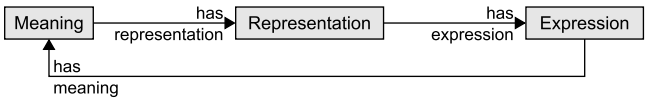


Figure 6: Transitive relationships in the SBVR meta-model

a *lowerBound*, and an *upperBound*. Therefore, an `EReference` maps to a `Port Definition`, with the *name* mapping to the `Port Name`, and the *lowerBound* and *upperBound* are mapped to the `Cardinality`. `Port Types` and their `Compatibility Definitions` are created for each `EReference` to ensure the associated ports can only connect to each other correctly.

When two `EObjects` are associated to one another through an `EReference` the appropriate `Port Individuals` of the equivalent `Component Individual` are *connected-to* each other.

Constraints

Arbitrary constraints in ECore are specified using annotations on model elements, written in some constraint language. These annotations are mapped to `Constraint Instances` and their corresponding `Constraint Expressions`. Special cases of constraints, particularly `Property Definition` and its sub-types (`Attribute Definition`, etc.), are utilised by previous mappings.

The constraints defined in our configuration model come from the SBVR specification. An example is shown in Figure 6 in that, the 'has meaning' relation between an 'Expression' and a 'Meaning' is allowed if and only if the 'Meaning' is connected to the 'Expression' through a 'Representation' and the 'has representation' and 'has expression' relations.

In the configurator used by our implementation the different aspects of the configuration knowledge are mapped to a generative CSP (or GCSP) as described in [Stumptner *et al.*, 1998]. This particular approach differs in its definition of some of the previously discussed concepts of [Soininen *et al.*, 1998] in the following respects:

- Non-leaf nodes in the taxonomy are assumed to be abstract; therefore, concrete component types that have sub-types are split into two types: an abstract super-type and a concrete sub-type.
- Ports are specified as necessary or optional; therefore, `Port Definitions` with higher cardinalities result in multiple ports that are then grouped into *port sets*, which allow quantitative reasoning over their members.

By configuring the SBVR models directly, we perform configuration of the concepts in a business specification rather than a syntactic parse tree as in [Kleiner *et al.*, 2009]. This results in an iteratively more detailed domain model, where new domain knowledge is taken into account each time a new sentence is processed. In addition, inconsistencies can be detected more easily than by reprocessing the model after new knowledge is added through a model transformation. Finally, the mapping to SBVR is simplified as the lexicon maps directly to the semantics, rather than an intermediate syntactic model. This solves the issue of multiple parse trees with correct mappings to SBVR as, for example, the parse of 'Each branch is included in exactly one local area' where 'local area' is considered a noun phrase would be inconsistent with the domain knowledge, while the parse where 'local area' is considered a noun would be consistent.

4 Experimental Results

We present the results of early experiments on the configuration of domain knowledge for NLP. We performed multiple tests of the example structural rule (1) and gathered statistics on the performance of the configurator in configuring these kinds of models. The results are summarised in Table 1.

The configurations produced were evaluated by hand for correctness. In each case the configuration was correct (corresponding with that shown in Figure 2b). The high number of variable assignments are due to relationships in the SBVR meta-model with cardinalities higher than one producing multiple ports in the configuration model; therefore, most port assignments are to the unconnected state.

It is interesting to note the correlation between the (minimum) number of backtracks and the number of components generated. This is due to the nature of the SBVR meta-model in two respects: (1) it contains a large number of relations with a cardinality of zero-to-many, and (2) it uses reified relations, which means that, in terms of configuration, each relation is represented as a component.

To prevent spurious connections between components, ports representing a zero-to-many relation are first set to the unconnected state; therefore, they are only connected to a component if being unconnected violates some constraint, causing a backtrack. Furthermore, the use of reified relations means that new *relation* components need to be created, even if it results in connecting two existing (non-relation) components. Therefore, in the optimal search of only connecting existing (non-relation) components, there will always be the same number of backtracks as generated components.

The higher number of backtracks in other configurations of the example are the result of the non-deterministic solver attempting variable assignments in a suboptimal order. This has a negative impact on performance. However, this could be avoided by providing SBVR specific procedural strategies for guiding the search [Stumptner *et al.*, 1998; Hotz and Wolter, 2013]. For example, ordering heuristics can be provided to change the order in which different component types or port types are assigned. Moreover, the search space could be reduced by preventing the creation of certain component types. For example, we assume a sentence is to be interpreted in the context of a provided vocabulary, hence we could prevent the

	Sentence (1)
# Constraints	197
# Variable Assignments	5162
Min. # Backtracks	6
Max. # Backtracks	25
Ave. # Backtracks	17
# Components Generated	6

Table 1: Performance statistics of the configuration process

creation of new object types, fact types, and other concepts related to the vocabulary aspects of the SBVR meta-model.

We are in the process of implementing a larger example, a portion of which assigned 4812 variables, generated 5 new components, and took 49 backtracks to do so. This emphasises the need for heuristics to help guide the search.

5 Related Work

Previous work in using configuration for natural language parsing has translated property grammars into a configuration model [Estratat and Henocque, 2004]. Using this approach, a simple context free-grammar ($a^n b^n$) and a subset of French were processed. This approach focuses primarily on the syntactic aspect of generating parse trees. Although the possibility of incorporating semantics is suggested in [Estratat and Henocque, 2004], none were incorporated in the processing of the natural language subset.

The approach of [Estratat and Henocque, 2004] was adapted to English and a Model-Driven Engineering environment in [Kleiner *et al.*, 2009]. Although their focus remains on the syntactic aspect of generating parse tress, SBVR semantics are partially taken into account by associating elements of the parse tree with SBVR types. This information is used to simplify the model transformation; however, the domain knowledge included in the SBVR model is not taken into account and, therefore, the process is not truly semantic.

Other approaches have used standard CSP translations of Dependency Grammars [Duchier, 1999] and, more recently, Property Grammars [Duchier *et al.*, 2012] in order to process natural language. However, both of these approaches focus on syntactic parsing, while we aim to incorporate semantics and domain knowledge directly into the parsing process.

6 Conclusions and Future Work

In this paper we have presented an approach to natural language processing that utilises configuration of domain knowledge to determine the validity of an expression. In effect, this performs natural language *understanding*, at least in terms of the semantic representation used. Moreover, we have demonstrated how techniques from Cognitive Linguistics can be combined with a translation of the SBVR meta-model (the semantic representation in our case) into a configuration problem in order to achieve this natural language understanding.

Our approach is novel in its combination of techniques from Cognitive Linguistics and configuration, and in that it performs configuration directly on the semantics of the domain knowledge. This is in contrast to previous approaches that use configuration or CSPs for natural language processing, as they tend to focus only on the syntactic aspect of generating a parse tree. As a result, our approach benefits from a simplified lexicon (important to our application in the business domain), improved mapping to the target semantics, and the semantic disambiguation of terms during processing.

The presented experimental results demonstrate the feasibility of our approach. In its current form, however, the configuration of the SBVR model can be inefficient and, therefore, future work will look at providing heuristics in order to ensure better performance in the configuration process. In addition, a more thorough evaluation of the process will be performed over larger examples in order to determine the effect of growing domain knowledge on the process.

References

[Duchier *et al.*, 2012] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, and Willy Lesaint. Property grammar parsing seen as a constraint optimization problem. In *Proc. Formal Grammar 2010/2011*, LNCS 7395, pages 82–96, 2012.

[Duchier, 1999] Denys Duchier. Axiomatizing dependency parsing using set constraints. In *Proc. Sixth Meeting on Mathematics of Language*, pages 115–126, 1999.

[Estratat and Henocque, 2004] Mathieu Estratat and Laurent Henocque. Parsing languages with a configurator. In *Proc. ECAI'2004*, volume 16, pages 591–595, 2004.

[Holmqvist, 1993] K. B. I. Holmqvist. *Implementing cognitive semantics: image schemata, valence accommodation and valence suggestion for AI and computational linguistics*. PhD thesis, Dept. of Cognitive Science Lund University, Lund, Sweden, 1993.

[Hotz and Wolter, 2013] Lothar Hotz and Katharina Wolter. Beyond physical product configuration configuration in unusual domains. *AI Communications*, 26:39–66, 2013.

[Kleiner *et al.*, 2009] M. Kleiner, P. Albert, and J. Bézivin. Configuring models for (controlled) languages. In *Proc. ConfWS'09*, pages 61–68, 2009.

[Langacker, 2008] R. W. Langacker. *Cognitive grammar: a basic introduction*. Oxford University Press, Oxford, New York, 2008.

[Nemuraite *et al.*, 2010] Lina Nemuraite, Tomas Skersys, Algirdas Sukys, Edvinas Sinkevicius, and Linas Ablonskis. VETIS tool for editing and transforming SBVR business vocabularies and business rules into UML&OCL models. In *Proc. ICIST 2010*, pages 377–384, 2010.

[OMG, 2006] OMG. *Meta Object Facility (MOF) Core Specification*. Object Management Group, 2006.

[OMG, 2008] OMG. *Semantics of Business Vocabulary and Business Rules (SBVR), v1.0*. Object Management Group, 2008.

[Soininen *et al.*, 1998] Timo Soininen, Juha Tiihonen, Tomi Männistö, and Reijo Solunen. Towards a general ontology of configuration. *AI EDAM*, 12(04):357–372, 1998.

[Stumptner *et al.*, 1998] Markus Stumptner, Gerhard E. Friedrich, and Alois Haselböck. Generative constraint-based configuration of large technical systems. *AI EDAM*, 12(04):307–320, 1998.

[Stumptner, 1997] Markus Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2):111–125, 1997.

[Sukys *et al.*, 2012] Algirdas Sukys, Lina Nemuraite, Bronius Paradauskas, and Edvinas Sinkevicius. Transformation framework for SBVR based semantic queries in business information systems. In *Proc. BUSTECH 2012*, pages 19–24, July 22-27 2012.

The effect of sales configurator capabilities on the value perceived by the customer through the customization process

Elisa Perin[1] and **Alessio Trentin** and **Cipriano Forza**
University of Padova, Italy
[1]perin@gest.unipd.it

Abstract

Literature has recently conceptualized five capabilities that a sales configurator should deploy in order to help avoid the product variety paradox, namely the risk that offering more product variety and customization to the market paradoxically results in a loss of sales. However, no studies have investigated the effect of such capabilities on the value that users derive from the experience of customizing their own products. To help narrow this research gap, in the present work we develop a number of hypotheses about the positive impact of such capabilities on the hedonic and creative value obtained by potential customers through the customization experience. We then test the hypothesized relationships and find empirical support for all of them.

1 Introduction

Sales configurators are software applications that support firms in identifying the complete and consistent commercial description of the product variant that best fits the customers' requirements among the company's offer [Forza and Salvador, 2008; Peng *et al.*, 2011]. The functions of a sales configurator include presenting the company's product space, meant as the set of products offered [Tseng and Piller, 2003], and preventing inconsistent or unfeasible solutions from being defined [Franke and Piller, 2003; Forza and Salvador, 2008].

Drawing upon prior research on sales configurators and customer decision processes, literature [Trentin *et al.*, 2013] has recently distilled five capabilities that a sales configurator should deploy in order to help avoid the product variety paradox. This is the risk that offering more product variety and customization to the customer, in an attempt to increase sales, paradoxically results in a loss of sales [Salvador and Forza, 2007].

However, no studies have analyzed the effect of these capabilities on the value that potential customers may derive from the experience of customizing their own products. Such a subjective value is posited by previous literature as increasing the customers' willingness to pay for mass-customized goods [Franke and Schreier, 2010; Franke *et al.*, 2010], and therefore it represents an important lever for mass customizers aiming at increasing their profitability. To help narrow this research gap, the present work develops and tests hypotheses about the positive impact of the abovementioned sales configurator capabilities on the value the customization experience provides to the potential customers.

2 Theoretical background and conceptual development

2.1 The value of the customization process

Consumer research has long recognized that shopping involves not only instrumental outcomes related to the merits of the goods or services acquired, but also experiential outcomes [Holbrook and Hirschman, 1982; Babin *et al.*, 1994]. The latter are emotional responses to the shopping experience that, when positive and rewarding, let customers obtain greater value from their shopping time [Holbrook and Hirschman, 1982; Babin *et al.*, 1994]. Greater perceived value, in turn, makes customers more willing to buy a product or pay a higher price for it [Baker *et al.*, 1992; Babin *et al.*, 1994; Franke and Schreier, 2010].

Experiencial value has been shown to influence customer's purchasing behaviour not only in the case of standard items, but also when products can be configured by using a Web-based sales configurator. Specifically, literature has unveiled that the value elicited by the configuration experience carry over to the evaluation of the self-configured product and increment the customer's willingness to pay [Franke and Schreier, 2010; Franke *et al.*, 2010; Merle *et al.*, 2010]. In particular, two types of experiencial values have been linked with the process of self-configuring a product, namely hedonic value and creative achievement value [Merle *et al.*, 2010].

Hedonic value

Hedonic value is defined as the value acquired from the experience's capacity to meet needs related to enjoyment, fun, or pleasure [Merle *et al.*, 2010]. In particular, with regard to a purchase situation, hedonic value reflects the consumers' appreciation for the shopping experience in

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

itself, regardless of any instrumental value of the purchased product [Babin *et al.*, 1994].

The importance of fulfilling the customer's needs for enjoyment, fun, or pleasure through the shopping experience has long been advocated by the marketing literature [e.g. Hirschman and Holbrook, 1982; Babin *et al.*, 1994; Childers *et al.*, 2001]. For example, literature has uncovered that instilling those feelings in the customer is a way to foster unplanned shopping decisions [Babin *et al.*, 1994], repurchase intentions [Jones *et al.*, 2006; Scarpi, 2012] or the use of online forms of shopping [Childers *et al.*, 2001].

Similar findings have also been reported in the mass-customization literature. Recent studies have uncovered that consumers configuring their own products are likely to experience process enjoyment [Franke and Schreier, 2010; Merle *et al.*, 2010]. These feelings can derive, for example, from learning one's own preferences by using the configuration process and/or from playing an active role in the design of a good [Franke and Schreier, 2010]. Noteworthy, these mechanisms are not inflenced by the characteristics of the products eventually configured, rather they result from the characteristics of the configuration process itself. For this reason the hedonic benefit is said to be "process-oriented" [Franke and Schreier, 2010].

Creative achievement value

Creative achievement value is defined as the value acquired by the customer from the feeling of accomplishment related to the creative task of codesigning [Merle *et al.*, 2010]. The elicitation of this type of value has also been referred to as the "I designed it myself" effect [Franke *et al.*, 2010]. Here the term "design" is used as including the configuration of a product within a predefined solution space [Franke *et al.*, 2010].

The concept of creative achievement value finds its theoretical support in the psychology literature. When people successfully complete a challenging task by their own efforts, they feel a positive emotion of self-reward, namely, pride [Weiner, 1985; Lea and Webley, 1997]. In other terms, when someone attains an outcome that signals his/her success in dealing with a challenge, s/he feels pride [Weiner, 1985; Franke *et al.*, 2010]. For example, when one does a complex Jigsaw puzzle, a favourable outcome of the process (i.e. having the puzzle completed) constitutes a positive feedback on one's own competences [Schreier, 2006]. This, in turn, gives the individual a strong feeling of pride for having done it oneself [Schreier, 2006].

The feeling of pride has also been studied with relation to the product customization task. The completion of such a task has been shown to give customers a sign of their competence and effectiveness in creating something, thus eliciting feelings of pride "of authorship" [Schreier, 2006]. This happens because, when faced with a configurable product instead of a standardized product, the customer perceives the shopping experience as being more difficult [Franke *et al.*, 2010]. Therefore, a favourable outcome to the configuration experience (i.e. a customized product that fits the customer's wants) embodies one's success in overcoming a challenge through the investment of personal efforts, time, and attention [Franke *et al.*, 2010]. As the favorableness of the outcome of the experience is a prerequisite for the user's perception of pride, the creative achievement benefit is said to be "output-oriented" [Schreier, 2006].

2.2 Sales configurator capabilities to improve customers' perceived value through the customization process

In the following subsections we argue that five capabilities, identified by previous research as key in avoiding the product variety paradox [Trentin *et al.*, 2013], also allow a sales configurator to increase the value perceived by a customer through the configuration process. These capabilities are: benefit-cost communication, user-friendly product-space description, easy comparison, flexible navigation, focused navigation capabilities (see Table 1).

Capability	Definition
Benefit-cost communication	The ability to effectively communicate the consequences of the available choice options both in terms of what the customer gets (benefits) and in terms of what the customer gives (monetary and nonmonetary costs)
User-friendly product-space description	The ability to adapt the product space description to the needs and abilities of different potential customers, as well as to different contexts of use
Easy comparison	The ability to minimize the effort required of a potential customer to compare previously created product configurations
Flexible navigation	The ability to minimize the effort required of a potential customer to modify a product configuration that he/she has previously created or is currently creating
Focused navigation	The ability to quickly focus a potential customer's search on a product space subset that contains the product configuration that best matches his/her idiosyncratic needs

Table 1: sales configurator capabilities (Trentin et al., 2013)

Impact of sales configurator capabilities on hedonic value

Benefit-cost communication capability

When a sales configurator has high benefit-cost communication capability, during the configuration task the customer is given pre-purchase feedbacks on the effects of the available choice options [Trentin *et al.*, 2013]. This is done, for example, by explaining what potential needs a given choice option contributes to fulfill and which is the price for such an option.

One of the product benefits customers are typically interested in is the aesthetic or, more in general, the sensorial aspect of the product s/he is considering for purchase [Li *et al.*, 2001; Fiore *et al.*, 2005]. A sales configurator with high benefit-cost communication capability is able to convey these sensorial aspects, for example through 360° product representation, the presence of sound recording, or virtual try-on technologies [Fiore *et al.*, 2005]. This allows customers to understand whether the

sensorial aspects of the configured product fit their needs. At the same time users are also allowed getting in closer contact with the company's offer through their senses, which is a need customers generally have while shopping [Hirschman and Holbrook, 1982]. When the shopping experience involves higher sensorial relation with products, the consumer's fantasy and imagination are stimulated suggesting elements of fun and playfulness [Jeong *et al.*, 2009]. This, in turn, increases the hedonic value that is perceived through the shopping experience [Shih, 1998; Fiore *et al.*, 2005; Jeong *et al.*, 2009].

Based on the above argument we posit that:

H1: The higher the level of benefit-cost communication capability deployed by a sales configurator, the higher the hedonic value perceived by the customer through the configuration process

User-friendly product-space description capability

When a sales configurator has high user-friendly product-space description capability, customers do not have to process product information that is not comprehensible for them [Alba and Lynch, 1997; Trentin *et al.*, 2013]. This is because the system adapts information contents according to their needs and abilities [Trentin *et al.*, 2013].

Since information content is customized based on one's needs and abilities, users perceive that the configuration process is up to their skills. Only when potential consumers perceive that a computer-mediated environment is congruent with their own skills can fun and enjoyment potentially occur [Hoffman and Novak, 1996]. Differently the consumers either become bored (i.e., their skills exceed the challenges) or anxious (i.e.. the challenges exceed their skills) [Hoffman and Novak, 1996].

Moreover when the customers are able to understand the product space characteristics, while using the sales configurator they learn about new products released in the market or new trends. Since learning about new products or trends is a source of enjoyment and entertainment for consumers [Childers *et al.*, 2001; Parsons, 2002; Arnold and Reynolds, 2003], this increases the hedonic value they perceive through the configuration experience.

Therefore, we posit that:

H2: The higher the level of user-friendly product-space description capability deployed by a sales configurator, the higher the hedonic value perceived by the customer through the configuration process

Easy comparison capability

When a sales configurator has high easy comparison capability, customers do not have to rely on their limited working memory to recover and compare configurations they have previously created [Trentin *et al.*, 2013]. This is because the system supports the retrieval of saved configurations and their comparison, for example through their side-by-side display [Trentin *et al.*, 2013].

The transformation of the decision from a memory-aided to a computer-aided process increases the number of product configurations that potential customers can explore and add to their consideration set, given their level of mental abilities or time availability [Alba and Lynch, 1997]. Decreased constraint to the exploration of the company's product space augments the users' feeling of freedom and spontaneity perceived during the configuration process. These feelings in turn drive the potential customer to obtain higher hedonic value out of the experience [Babin *et al.*, 1994].

Based on the above argument we posit that:

H3: The higher the level of easy comparison capability deployed by a sales configurator, the higher the hedonic value perceived by the customer through the configuration process

Flexible navigation capability

When a sales configurator has high flexible navigation capability, customers can quickly make and undo changes to a current configuration or to previously created ones. This can be done, for example, through the use of bookmarks that redirect to previous steps of the configuration process [Randall *et al.*, 2005; Trentin *et al.*, 2013].

As going back to previous steps of the configuration is easier, the potential customer can conduct many trial-and-error tests to evaluate the effects of different choices made available by the company [Trentin *et al.*, 2013]. In this way, the exploration of the solution space is pursued more actively by the customer, compared to cases where excessive time/mental resources demands discourage customer's non-linear movements through the solution space. A more active role, in turn, makes the potential customer perceive the process as an exciting play, thus fulfilling his/her need for enjoyment and fun [Babin *et al.*, 1994; Arnold and Reynolds, 2003; To *et al.*, 2007].

Based on the above, we posit that:

H4: The higher the level of flexible navigation capability deployed by a sales configurator, the higher the hedonic value perceived by the customer through the configuration process

Focused navigation capability

A sales configurator with focused navigation capability does not force potential customers to go through and evaluate a number of product options that they regard as certainly inappropriate for themselves [Trentin *et al.*, 2013]. A way to do this is, for example, to provide starting points, that is, product configurations that are close to the customer's ideal solution and that may be further customized to meet customer's needs more accurately [Trentin *et al.*, 2013].

The restriction of the search only to a limited set of product solutions that are of interest to the customer, increases the likelihood that s/he soon finds something that raises his/her attention and engagement. This, in turn, leaves more time to the person to focus on what is more engaging and stimulating for him/her, thus increasing the enjoyment perceived during the configuration process.

Therefore, we posit that:

H5: The higher the level of focused navigation capability deployed by a sales configurator, the higher the hedonic value perceived by the customer through the configuration process

Impact of sales configurator capabilities on creative value

Benefit-cost communication capability

By delivering pre-purchase feedback on the effects of the available choice options, a sales configurator with high benefit-cost communication capability allows potential customers to understand the value that they can derive from these options [Trentin *et al.*, 2013]. The learning process enabled by such a capability makes a potential customer more confident that the product configuration s/he has selected is the one that best fits her/his needs within the company's product space [Trentin *et al.*, 2013]. In other terms, a configurator with high benefit-cost communication capability makes the customers feel they have obtained the most favorable outcome out of the configuration process and out of the efforts that they have invested in such a process. As pride arises when it is possible to attribute a favorable outcome to the self [Weiner, 1985], the benefit-cost communication capability has a role in augmenting the feeling of pride perceived by the users through configuring their own products. This feeling, in turn increases the creative achievement value that the customer derives from the customization process [Merle *et al.*, 2010].

Based on the above arguments, we posit that:

H6: The higher the level of benefit-cost communication capability deployed by a sales configurator, the higher the creative value perceived by the customer through the configuration process

User-friendly product-space description capability

By tailoring both information content and information format to the abilities of different potential customers, a sales configurator deploying user-friendly product-space description capability facilitates the users' understanding of the solution space characteristics [Trentin *et al.*, 2013]. Without such understanding, it would be difficult for the customer to complete the configuration task and obtain a product configuration that corresponds to one's expectations and needs [Fürstner *et al.*, 2012; Trentin *et al.*, 2013]. This, in turn, would make the customer attribute a negative outcome to the efforts employed in the process. Conversely, when potential customers, supported by the user-friendly product-space description capability, are able to obtain the needed products, they feel "smarter" than their counterparts (co-workers, neighbors, relatives). This is because they are able to co-designed a product instead of buying something created by somebody else [Schreier, 2006]. This makes them feel pride of authorship, and increses the creative achievement value derived from the process [Schreier, 2006; Merle *et al.*, 2010].

Based on the above arguments, we posit that:

H7: The higher the level of user-friendly product-space description capability deployed by a sales configurator, the higher the creative value perceived by the customer through the configuration process

Easy comparison capability

By enabling the comparison between previously created configurations, a sales configurator deploying easy comparison capability fosters the users' learning about the instrumental value they would derive from the product being configured. This is because, in assessing the value of a particular product solution, customers tend to rely on comparisons with other product alternatives [Simonson and Tversky, 1992; Simonson, 2005]. The learning process enabled by easy comparison capability makes a potential customer more confident that s/he is selecting the product configuration that best fits his/her needs [Trentin *et al.*, 2013]. As pride arises when a favorable outcome is ascribed to one's contribution [Weiner, 1985], higher easy comparison capability augments the feeling of pride perceived by the user through configuring their product.

Moreover, the possibility to compare previously saved configurations relieves the customer from manually or mentally recording relevant information (e.g., design parameters and product attributes) of the previously chosen configurations [Randall *et al.*, 2005]. In this way, the customer's mental abilities, or the time availability for manually recording information, become less salient and s/he is enabled to configure a higher number of products. By being able to configure a higher number of products, the customer can give free reins to his/her creativity, exploring multiple combinations of product features (for example different combinations of colors). This provides more chances for the evaluation of one's creative skills, and thus for eliciting pride feelings [Harter, 1985]. Pride, in turn, increases the creative achievement value that the customer derives from the customization process [Merle *et al.*, 2010].

Therefore, we posit that:

H8: The higher the level of easy comparison capability deployed by a sales configurator, the higher the creative value perceived by the customer through the configuration process

Flexible navigation capability

By enabling potential customers to quickly make and undo changes to previously created product configurations, a sales configurator with high flexible navigation capability enables users to conduct more trial-and-error tests to evaluate the effects of available choices [Trentin *et al.*, 2013]. This experimentation promotes potential customers' learning about the value they would derive from the product being configured. Such learning process makes potential customers more confident that the product configuration they have selected is the one that best fits their needs within the company's product space [Trentin *et al.*, 2013]. As the potential customers feel they have obtained the most favorable outcome out of the configuration process, they

feel proud of their accomplishment, which can be attributed to their own efforts [Weiner, 1985].

Moreover, as the users are able to conduct many trial-and-error tests, they can give free reins to their creativity, by exploring more combinations of product features. This, in turn, provides more chances for evaluating one's creative competences. As pride is a positive, self-rewarding emotion arising from the evaluation of one's competence [Harter, 1985; Schreier, 2006], a sales configurator with flexible navigation capability is likely to make the users experience stronger feelings of pride. This in turn increases the creative achievement value they obtain [Merle *et al.*, 2010].

Therefore, we posit that:

> *H9: The higher the level of flexible navigation capability deployed by a sales configurator, the higher the creative value perceived by the customer through the configuration process*

Focused navigation capability

A sales configurator with focused navigation capability prevents potential customers from going through a number of product options that they regard as certainly inappropriate for themselves [Trentin *et al.*, 2013]. As the size of their search problem is reduced, potential customers can spend more time and effort in exploring the product options for which their preferences are less certain. In addition, they can rely on more time-consuming, compensatory decision strategies for the resolution of between-attribute conflicts [Bettman *et al.*, 1990]. This makes them more confident that the chosen solution is the one that best fits their needs within the company's product space. As a consequence, the potential customers feel they have obtained an outcome that is really up to their personal capacities, rather than a sub-optimum obtained under time-constraints, and they are more likely to feel proud of themselves. Pride, in turn increases the creative achievement value that the potential customers derive from the customization process [Merle *et al.*, 2010].

Based on the above arguments, we posit that:

> *H10: The higher the level of focused navigation capability deployed by a sales configurator, the higher the creative value perceived by the customer through the configuration process*

3 Method

To test our hypotheses we conducted an empirical analysis using survey data collected from a sample of 675 sales configuration experiences made by 75 students at the authors' university (age range: 24-27; 30% females, mean expertise in using Internet to conduct transactions[1]: 3.95, standard deviation: 1.90). Each participant was asked to configure a product, according to his/her individual needs, on nine Web-based sales configurators for consumer goods and to fill out a questionnaire for each experience. In this questionnaires, participants had to rate the capabilities of each configurator and the level of hedonic and creative value they had derived from the configuration process. The items used to measure these constructs are reported in Appendix A.

The chosen data analysis method is the structural equation modeling, using LISREL 8.80. Following Anderson and Gerbing [1988], we decided to adopt a two-step approach, assessing construct validity before the simultaneous estimation of the measurement and structural models. Moreover, since our variables did not meet the assumption of multivariate normal distribution (Mardia's test significant at $p<0.001$) we applied the Satorra-Bentler correction to produce robust maximum likelihood estimates of standard errors and Chi-square.

4 Results

Prior to conducting the analysis, we decided to control for possible effects of participants' characteristics. Consequently, and consistent with prior studies [Liu *et al.*, 2006; Trentin *et al.*, 2013], we regressed our observed indicators on 75 dummies representing the participants in our study and used the standardized residuals from this linear, ordinary least square regression model as our data in all the subsequent analyses.

Confirmatory factor analysis (CFA) was subsequently employed to assess unidimensionality, convergent validity, discriminant validity, and reliability of our measurement scales. A CFA model specifies the posited relations of the observed variables to the underlying latent constructs, with these constructs allowed to correlate freely [Anderson and Gerbing, 1988]. Our CFA model showed good fit indices (RMSEA (90% CI)= 0.0576 (0.0531; 0.0623), Satorra-Bentler Scaled $\chi^2/df(df)$ = 2.80 (231), CFI=0.990, NFI=0.984), meaning that our hypothesized factor structure reproduced the sample data well.

The standardized factor loadings (S.F.L, see in Appendix A) were all in their anticipated direction, greater than 0.50 and statistically significant at $p<0.001$. Altogether, these results suggested unidimensionality (a set of empirical indicators reflect one, and only one, underlying latent factor) and good convergent validity (the multiple items used as indicators of a construct significantly converge) of our measurement scales [Campbell and Fiske, 1959; Anderson and Gerbing, 1988].

Discriminant validity, which measures the extent to which the individual items of a construct are unique and do not measure other constructs, was tested using Fornell and Larcker's [1981] procedure. For each latent construct, the square root of the average variance extracted (AVE) exceeded the correlation with all the other latent variables. This suggests that our measurement scales represent distinct latent variables [Fornell and Larcker, 1981].

[1] measured as in [Hernández et al 2010], on a seven-point Likert scale (7 = completely agree, 1 = completely disagree). Only one factor with eigenvalue higher than 1 was extracted, with a principal component analysis, 85% variance explained by this factor, Cronbach's alfa: 0.94.

Reliability of the measurement scale was assessed using both AVE and the Werts, Linn, and Joreskog (WLJ) composite reliability (C.R.) method [Werts *et al.*, 1974]. All the WLJ composite reliability values were greater than 0.70 and all the AVE scores largely exceeded 0.50 (see Appendix A). This indicates that a large amount of the variance is captured by each latent construct rather than due to measurement error [Fornell and Larcker, 1981; O'Leary-Kelly and J. Vokurka, 1998].

Finally, we examined the measurement model complemented by the structural paths corresponding to our hypotheses. All five sales configurator capabilities are posited as helping firms increasing the hedonic and creative value perceived by their potential customer through the configuration experience. Accordingly, these capabilities were restricted to impact both hedonic value and creative value. Results show that all the path coefficients of the estimated model are positive and statistically significant, indicating that all our hypotheses are supported. Table 2 reports the Lisrel estimates of the path coefficients, with standard errors in brackets.

	BCC	EC	UFDC	FlexN	FocN
HE	0.221 *(0.086*)*	0.102 *(0.037**)*	0.151 *(0.067*)*	0.283 *(0.065***)*	0.502 *(0.088***)*
CA	0.150 *(0.085§)*	0.166 *(0.035***)*	0.137 *(0.066*)*	0.267 *(0.055***)*	0.261 *(0.082***)*

Table 2: path coefficients of the estimated model

Significant at: *** p < 0.001; ** p < 0.01; * p < 0.05; § p < 0.10; BCC = benefit-cost communication; EC= easy comparison; UFD= user-friendly product-space description; FlexN= flexible navigation; FocN=focused navigation; HE= hedonic value; CA= creative achievement value

5 Conclusion

The present paper has developed and tested hypotheses about the positive impact of five sales configuration capabilities on the hedonic value and the creative value perceived by users through the customization process. These capabilities are: focused navigation, flexible navigation, easy comparison, benefit-cost communication, and user-friendly product-space description capabilities [Trentin *et al.*, 2013].

By finding empirical support for the hypothesized relationships between such sales configurator capabilities and the value provided by a configuration process, this work adds to the debate surrounding information technology support to mass customization [e.g. Blecker and Friedrich, 2007; Forza and Salvador, 2008]. Mass customization involves not only improving compatibility between product customization and the firm's operational performance, but also augmenting the value of the customization as perceived by the customer [Franke and Schreier, 2010; Franke *et al.*, 2010; Merle *et al.*, 2010]. The results of this study improve our understanding of how product configurators should be designed to foster such a value, which is a way for mass customizers to increase customers' willingness to pay for a customized product [Franke and Schreier, 2010; Franke *et al.*, 2010], and thus to increase the value of a mass customization strategy.

Acknowledgements

We acknowledge the financial support of the University of Padova, Project ID CPDA109359.

Appendix A

Sales configurator capabilities[(a)]

Benefit-cost communication capability (AVE: 0.697; C.R.: 0.873):

- BCC1 Thanks to this system, I understood how the various choice options influence the value that this product has for me (S.F.L.: 0.858, P<0.001).
- BCC2 Thanks to this system, I realized the advantages and drawbacks of each of the options I had to choose from (S.F.L.: 0.792, P<0.001).
- BCC3 This system made me exactly understand what value the product I was configuring had for me (S.F.L.: 0.853, P<0.001).

Easy comparison capability (AVE: 0.796; C.R.: 0.939):

- EC1 The system enables easy comparison of product configurations previously created by the user (S.F.L.: 0.894, p<0.001).
- EC2 The system lets you easily understand what previously created configurations have in common (S.F.L.: 0.948, p<0.001).
- EC3 The system enables side-by-side comparison of the details of previously saved configurations (S.F.L.: 0.807, p<0.001).
- EC4 The systems lets you easily understand the differences between previously created configurations (S.F.L.: 0.913, p<0.001).

User-friendly product-space description capability (AVE: 0.730; C.R.: 0.890):

- UFDC1 The system gives an adequate presentation of the choice options for when you are in a hurry, as well as when you have enough time to go into the details (S.F.L.: 0.883, p<0.001).
- UFDC2 The product features are adequately presented for the user who just wants to find out about them, as well as for the user who wants to go into specific details (S.F.L.: 0.907, p<0.001).
- UFDC3 The choice options are adequately presented for both the expert and inexpert user of the product (S.F.L.: 0.766, p<0.001).

Flexible navigation capability (AVE: 0.614; C.R.: 0.826):

- FlexN1 The system enables you to change some of the choices you have previously made during the configuration process without having to start it over again (S.F.L.: 0.738, p<0.001).
- FlexN2 With this system, it takes very little effort to modify the choices you have previously made during the configuration process (S.F.L.: 0.788, p<0.001).
- FlexN3 Once you have completed the configuration process, this system enables you to quickly change any

choice made during that process (S.F.L.: 0.822, p<0.001).

Focused navigation capability (AVE: 0.724; C.R.: 0.913):

FocN1 The system made me immediately understand which way to go to find what I needed (S.F.L.: 0.857, p<0.001).

FocN2 The system enabled me to quickly eliminate from further consideration everything that was not interesting to me at all (S.F.L.: 0.790, p<0.001).

FocN3 The system immediately led me to what was more interesting to me (S.F.L.: 0.893, p<0.001).

FocN4 This system quickly leads the user to those solutions that best meet his/her requirements (S.F.L.: 0.860, p<0.001).

Perceived benefits of mass customization from a consumer viewpoint[(b)]

Hedonic value (AVE: 0.882; C.R.: 0.957):

HE1 I found it fun to customize this product (S.F.L.: 0.952, p<0.001).

HE2 Configuring this product was a really gratifying thing to do (S.F.L.: 0.908, p<0.001).

HE3 Customizing this product was a real pleasure(S.F.L.: 0.956, p<0.001).

Creative achievement value (AVE: 0.757; C.R.: 0.925):

CA1 I see myself as the author of the product which I configured (S.F.L.: 0.913, p<0.001).

CA2 I felt really creative while configuring this product (S.F.L.: 0.913, p<0.001).

CA3 The company gave me a lot of freedom while creating this product (S.F.L.: 0.913, p<0.001).

CA4 By personalizing this product, I had the impression of creating something (S.F.L.: 0.877, p<0.001).

[(a)] Trentin et al 2013 ; [(b)] Merle et al. 2010, adapted

References

[Alba and Lynch, 1997] Joseph Alba and John Lynch. Interactive Home Shopping: Consumer, Retailer, and Manufacturer Incentives to Participate in Electronic Marketplaces. *Journal of Marketing,* 61(3): 38-53, 1997.

[Anderson and Gerbing, 1988] James C. Anderson and David W. Gerbing. Structural Equation Modeling in Practice: A Review and Recommended Two-Step Approach. *Psychological Bulletin,* 103(3): 411-423, 1988.

[Arnold and Reynolds, 2003] Mark J. Arnold and Kristy E. Reynolds. Hedonic shopping motivations. *Journal of Retailing,* 79(2): 77-95, 2003.

[Babin *et al.*, 1994] Barry J. Babin, William R. Darden, and Mitch Griffin. Work and/or Fun: Measuring Hedonic and Utilitarian Shopping Value. *Journal of Consumer Research,* 20(4): 644-656, 1994.

[Baker *et al.*, 1992] Julie Baker, Michael Levy, and Dhruv Grewal. An experimental approach to making retail store environmental decisions. *Journal of Retailing,* 68(4): 445-460, 1992.

[Bettman *et al.*, 1990] James R. Bettman, Eric J. Johnson, and John W. Payne. A componential analysis of cognitive effort in choice. *Organizational Behavior and Human Decision Processes,* 45(1): 111-139, 1990.

[Blecker and Friedrich, 2007] Thorsten Blecker and Gerhard Friedrich. *Mass Customization Information Systems in Business*. IGI Global, London, UK, 2007.

[Campbell and Fiske, 1959] Donald T. Campbell and Donald W. Fiske. Convergent and discriminant validation by the multitrait-multimethod matrix. *Psychological Bulletin,* 56(2): 81-105, 1959.

[Childers *et al.*, 2001] Terry L. Childers, Christopher L. Carr, Joann Peck, and Stephen Carson. Hedonic and utilitarian motivations for online retail shopping behavior. *Journal of Retailing,* 77(4): 511-535, 2001.

[Fiore *et al.*, 2005] Ann Marie Fiore, Jihyun Kim, and Hyun-Hwa Lee. Effect of image interactivity technology on consumer responses toward the online retailer. *Journal of Interactive Marketing,* 19(3): 38-53, 2005.

[Fornell and Larcker, 1981] Claes Fornell and David F. Larcker. Evaluating Structural Equation Models with Unobservable Variables and Measurement Error. *Journal of Marketing Research,* 18(1): 39-50, 1981.

[Forza and Salvador, 2008] Cipriano Forza and Fabrizio Salvador. Application support to product variety management. *International Journal of Production Research,* 46(3): 817-836, 2008.

[Franke and Piller, 2003] Nikolaus Franke and Frank T. Piller. Key Research Issues in User Interaction with Configuration Toolkits in a Mass Customization System. *International Journal of Technology Management,* 26(5-6): 578–599, 2003.

[Franke and Schreier, 2010] Nikolaus Franke and Martin Schreier. Why Customers Value Self-Designed Products: The Importance of Process Effort and Enjoyment. *Journal of Product Innovation Management,* 27(7): 1020-1031, 2010.

[Franke *et al.*, 2010] Nikolaus Franke, Martin Schreier, and Ulrike Kaiser. The "I Designed It Myself" Effect in Mass Customization. *Management Science,* 56(1): 125–140, 2010.

[Fürstner *et al.*, 2012] Igor Fürstner, Zoran Anišić, and Márta Takács. Product Configurator Self-Adapting to Different Levels of Customer Knowledge. *Acta Polytechnica Hungarica,* 9(4): 129-150, 2012.

[Harter, 1985] Susan Harter. Competence as a dimension of self-evaluation: towards a comprehensive model of self-worth. In R. Leahy (Ed.), *The Development of the Self*, pages 55–121.Academic Press, New York, 1985.

[Hernández et al 2010] B. Hernandez, J. Jimenez, and M. J. Martin. Customer behavior in electronic commerce: The moderating effect of e-purchasing experience. *Journal of Business Research*, 63(9-10): 964-971, 2010.

[Hirschman and Holbrook, 1982] Elizabeth C. Hirschman and Morris B. Holbrook. Hedonic consumption: emerging concepts, methods and propositions. *The Journal of Marketing,* 46(3): 92-101, 1982.

[Hoffman and Novak, 1996] Donna L. Hoffman and Thomas P. Novak. Marketing in hypermedia computer-mediated environments: Conceptual foundations. *Journal of Marketing,* 60(3): 50, 1996.

[Holbrook and Hirschman, 1982] Morris B. Holbrook and Elizabeth C. Hirschman. The Experiential Aspects of Consumption: Consumer Fantasies, Feelings, and Fun. *Journal of Consumer Research,* 9(2): 132-140, 1982.

[Jeong *et al.*, 2009] So Won Jeong, Ann Marie Fiore, Linda S. Niehm, and Frederick O. Lorenz. The role of experiential value in online shopping: The impacts of product presentation on consumer responses towards an apparel web site. *Internet Research,* 19(1): 105-124, 2009.

[Jones *et al.*, 2006] Michael A. Jones, Kristy E. Reynolds, and Mark J. Arnold. Hedonic and utilitarian shopping value: Investigating differential effects on retail outcomes. *Journal of Business Research,* 59(9): 974-981, 2006.

[Lea and Webley, 1997] Stephen E. G. Lea and Paul Webley. Pride in economic psychology. *Journal of Economic Psychology,* 18(2-3): 323-340, 1997.

[Li *et al.*, 2001] Hairong Li, Terry Daugherty, and Frank Biocca. Characteristics of virtual experience in electronic commerce: A protocol analysis. *Journal of Interactive Marketing,* 15(3): 13-30, 2001.

[Liu *et al.*, 2006] Gensheng Liu, Rachna Shah, and Roger G. Schroeder. Linking Work Design to Mass Customization: A Sociotechnical Systems Perspective. *Decision Sciences,* 37(4): 519-545, 2006.

[Merle *et al.*, 2010] Aurélie Merle, Jean-Louis Chandon, Elyette Roux, and Fabrice Alizon. Perceived Value of the Mass-Customized Product and Mass Customization Experience for Individual Consumers. *Production and Operations Management,* 19(5): 503-514, 2010.

[O'Leary-Kelly and Vokurka, 1998] Scott W. O'Leary-Kelly and Robert J. Vokurka. The empirical assessment of construct validity. *Journal of Operations Management,* 16(4): 387-405, 1998.

[Overby and Lee, 2006] Jeffrey W. Overby and Eun-Ju Lee. The effects of utilitarian and hedonic online shopping value on consumer preference and intentions. *Journal of Business Research,* 59(10/11): 1160-1166, 2006.

[Parsons, 2002] Andrew G. Parsons. Non-functional motives for online shoppers: why we click. *Journal of Consumer Marketing,* 19(5): 380-392, 2002.

[Peng *et al.*, 2011] David Xiaosong Peng, Gensheng Jason Liu, and Gregory R. Heim. Impacts of Information Technology on Mass Customization Capability of Manufacturing Plants. *International Journal of Operations & Production Management,* in press, 2011.

[Randall *et al.*, 2005] Taylor Randall, Christian Terwiesch, and Karl T. Ulrich. Principles for User Design of Customized Products. *California Management Review,* 47(4): 68-85, 2005.

[Salvador and Forza, 2007] Fabrizio Salvador and Cipriano Forza. Principles for efficient and effective sales configuration design. *International Journal of Mass Customisation,* 2(1-2): 114-127, 2007.

[Scarpi, 2012] Daniele Scarpi. Work and Fun on the Internet: The Effects of Utilitarianism and Hedonism Online. *Journal of Interactive Marketing,* 26(1): 53-67, 2012.

[Schreier, 2006] Martin Schreier. The value increment of mass-customized products: an empirical assessment. *Journal of Consumer Behaviour,* 5(4): 317-327, 2006.

[Shih, 1998] Chuan-Fong Shih. Conceptualizing consumer experiences in cyberspace. *European Journal of Marketing,* 32(7): 655-663, 1998.

[Simonson, 2005] Itamar Simonson. Determinants of customers' responses to customized offers: conceptual framework and research propositions. *Journal of Marketing,* 69(1): 32-45, 2005.

[Simonson and Tversky, 1992] Itamar Simonson and Amos Tversky. Choice in context: tradeoff contrast and extremeness aversion. *Journal of Marketing Research,* 29(3): 281-295, 1992.

[To *et al.*, 2007] Pui-Lai To, Chechen Liao, and Tzu-Hua Lin. Shopping motivations on Internet: A study based on utilitarian and hedonic value. *Technovation,* 27(12): 774-787, 2007.

[Trentin *et al.*, 2013] Alessio Trentin, Elisa Perin, and Cipriano Forza. Sales configurator capabilities to avoid the product variety paradox: Construct development and validation. *Computers in Industry,* 64(4): 436–447, 2013.

[Tseng and Piller, 2003] Mitchell M. Tseng and Frank T. Piller. *The Customer Centric Enterprise: Advance in Mass Customization and Personalization*, Berlin, Germany, 2003.

[Weiner, 1985] Bernard Weiner. An attributional theory of achievement motivation and emotion. *Psychological Review,* 92(4): 548-573, 1985.

[Werts *et al.*, 1974] Charles E. Werts, Robert L. Linn, and Karl G. Jöreskog. Intraclass Reliability Estimates: Testing Structural Assumptions. *Educational and Psychological Measurement,* 34(1): 25-33, 1974.

Generation of predictive configurations for production planning

Tilak Raj Singh
Production Tools (IT)
Mercedes-Benz R&D Bangalore
tilak.singh@daimler.com

Narayan Rangaraj
Industrial Eng. & Operations Research
IIT Bombay, Mumbai
narayan.rangaraj@iitb.ac.in

Abstract

We provide a production planning framework for variant rich customized products (such as automobiles, computers), by calculating entirely constructible configuration sets for future customer demands in a novel manner. Most of the established approaches analyse configurations out of historical order banks for estimating the appropriate set of future demands. In the current environment of rapidly changing designs and highly customized products, historical demands cannot easily be extrapolated to capture future market demand and may not even retain future product document restrictions. In this paper, our aim is to generate configuration sets such that (1) they represent customers buying behaviour (derived from configurations produced in the past and sales planning at aggregate level) (2) they are consistent with the product documentation. Configuration generation is formulated as guided search procedures which utilize the Satisfiability framework. Selection of configuration sets for planning is done by a large scale optimization model. We use column generation and other techniques to solve this large scale optimization model.

1 Introduction

In the customer focused order-fulfilment strategies such as Built-To-Order and Assemble-To-Order, mid to long term (6 months - 3 years) planning activities in production and logistics are supported with aggregate level of forecast from sales and marketing. Through sales forecast it is possible to get estimate of total volume for entire product line. In addition to this we get demand estimates for key attributes of the product [Srinivasan and Swaminathan, 1997]. For example, in case of automotive, attributes can be engine type, body style, air condition, and so forth. During the estimation demand characteristics of future customers, the dependencies between attributes and components provided by designers or customers may not have been considered [Olsen and Saetre, 1997]. Component dependencies by design can be found in product documentation and these will be reflected in the Bill-Of-Material system [Kaiser and Küchlin, 2001]. Dependencies from customer point of view may not be straightforward but these can be extracted from variants produced in the past. The important thing to note is that these dependencies change with continuous changes in the design (introduction of new feature, parts or components) and because of changing market, legislation and economic conditions.

Starting from sales planning inputs, the primary task of the production program planning activities is to know which parts and components need to be available at what time and in what amount, in order to produce the planned product units efficiently? This has to be done even when the company has not received any real customer orders.

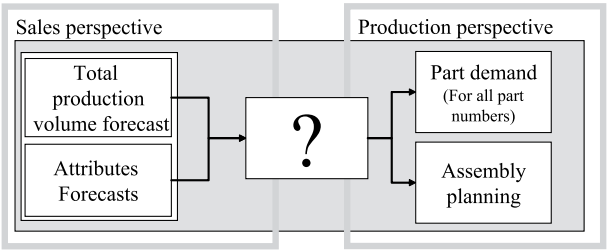


Figure 1: Need of methods for consistent transformation of information between sales and production planning

The derivation of part demand or workload at any assembly station may not be straightforward from the sales planning inputs. For example, one might get information from sales forecast that attribute parking assistance system and automatic lane departure system will be used in 50% and 40% of the cars respectively. This information may not be enough for calculation of the demand for a specific steering wheel. Selection of steering wheel may depend on if both attribute are selected together or individually. Assume that there is one steering wheel that will be used if both the above attributes are selected in the same configuration. From the sales planning as an independent forecast for attributes, the demand of this specific steering wheel may lie between 0% to 40%. Consequently, without additional assumptions, we cannot determine demands of all parts required for order fulfilment.

As shown in Figure 1 there is a need to build a medium which can transfer consistent information across various departments involved in the customer order processing to enable better program planning. One way to achieve this is by planning with fully specified customer configurations. As cus-

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

tomer configurations are not available for mid to long term planning, most manufacturers use variants produced in the past to estimate the appropriate set of future demands. Due to the introduction of engineering changes and shifts in market expectation, variants produced in the past become statistically less significant to capture future customers demand characteristics. Also, engineering changes in the product make some configurations obsolete and cannot be re-produced exactly. Thus, simple extrapolation of historical demands to capture future demands characteristics may cause information distortion in the early stage of the production planning.

Our main aim in this paper is to provide an automated framework to supplement the historical configurations set in such a way so that the underlying configurations set can be more relevant to capture future demand characteristics. Once the desired configurations are generated we will select a target number of configurations to propose a production plan. The main challenge in defining future configurations of the product is that the solution space is huge (an enormous number of configurations are technically feasible). We will propose an integrated configurations generation and selection approach that will calculate only a few (as compared to the full solution space) configurations to complete the set of base configurations set for planning. In our work, we will argue that by considering sales estimates, engineering dependencies, production restriction and customer buying behaviour during configuration generation, the number of valid configurations can reduce significantly.

The rest of the paper is structured in following way: We will present a review of literature in Section 2 to motivate the need and use of product configurations in various planning activities across the organization. In Section 3 we will present formal problem descriptions with input data and their characteristics. In Section 4 we will discuss a heuristic for generating configurations directly from product documentation (i.e. the engineering document). The mathematical model building and solution methodology will be discussed in Section 5. Section 6 will describe initial computational experiments on industry size examples.

2 Literature Review

To manage product variety in mass customization techniques such as product differentiation and postponement are well studied approaches, offering flexible manufacturing for high variety product [Harrison *et al.*, 2004]. The basic idea is to design and manufacture the product in such a way so that variety can be introduced at the last stage of the production. The partially assembled standard products are produced till the point no differentiation is required. Final assembly is done based on customer configuration by adding specific product features. The work in progress (WIP) inventories are maintained to offer customer attractive lead time with required variety [Swaminathan and Tayur, 1998]. For a manufacturer who follows lean manufacturing or Just-in-Time (JIT) approaches, any kind of inventory either of individual parts or components or as a WIP is highly undesirable. As product technology and design changes continuously with respect to time, it might be difficult and costly to introduce variety at the end of the production.

Product configuration system has been a key enabler of mass customization by capturing the customer demand in most consistent way. Although, initial focus of configuration system was to provide significant reduction in customer order response time by enabling customer-product interface [Salvador and Forza, 2004]. As the customer order fulfilment process varies based on the product configurations, there is a need to utilize configurations technique in various planning and process design [Aldanondo and Vareilles, 2008]. Product configurations act as a medium to translate information between customer, sales, manufacturing and other supply chain players. For example maintaining consistent bill-of-material, or finding range of product with certain characteristics [Astesana *et al.*, 2010].

By utilizing product configurations in the early stage of planning hybrid order fulfillment strategies such as Virtual-Build-to-Order (VBTO) system can be created [Brabazon and MacCarthy, 2004]. The fundamental capability for a VBTO system is the ability to search the order fulfilment pipeline on behalf of the customer. These virtual (not created by end customer) configurations can be reconfigured with respect to actual customer configuration with minimum difference from customer preference. At the end, efficiency of systems such as VBTO mainly depends upon the correlation between the planned and real orders. If we are able to simulate configurations according to customers need, then we will get high level of satisfaction and smooth processes in customer order fulfilment.

It has been agreed in literature that efficient configuration system which co-ordinates and covers information from all available sources (e.g. sales, marketing, assembly, logistics, design, and customers) leads to significant gain in customer order fulfilment process [Trentin *et al.*, 2011]. However, most of the efforts in the past are devoted to use product configurations for reducing the lead time and maintaining customers buying behaviour. The generation of configurations has received considerably less attention. Hayler [Hayler, 1999] developed a sequential procedure for generating product configuration from rule based system. Products attribute classes (levels) are created and each virtual configuration selects attribute based on its forecast rates. The approach can be compared as a product configurator system. A product configurator is created form rule based design document. Selection of attribute from each step of configuration is supported by attributes selection rate, historical orders, association rules and experts experience. These permutation procedures often hamper the result quality and require manual intervention to match desired output. Stautner [Stautner, 2001] discussed configuration generation problem by identifying configurations form recent history through cluster analysis. Historical orders are modified in such a way so that it can fit in future planning requirements. The method involves manual procedure to create final configurations. In case of new product such as electric or hybrid cars which open new market segments for manufacturer and does not have customer history. The current available methods find difficulties in generating future product configuration which matches given input requirements from design, sales and production.

3 The planning problem

The major aim of this paper is the development of an automated procedure that supports production planning, part demand calculation and capacity management for the short-term as well as for a medium-term planning horizon up to two years from the start of production.

As discussed in the introduction, in a customer oriented production environment, planning may be done at an aggregate level such as modules or attributes. One problem in this method is that engineering constraints between attributes may not be taken into consideration. For example, selection of front bumper in the car may depend on body style type, headlamp, and optional feature and sensor mounted on the bumper. Drawing estimates of future bumper demand without consideration of such dependencies may give unreliable estimates.

This problem can be avoided if planning is based on complete products. However, drawing a small number of representative configurations from an enormously large set of possible configurations is a daunting task. One way to attack this problem is by utilizing customers demand in the past. Customer order history is an important input for capturing customer buying behaviour, required for future planning activities. In Figure 2 the planning tasks related to logistics and assembly are derived through extrapolation of configurations produced in the past. Once the fully specified configurations are known, assembly related processes can be optimized for the selected production program. From the logistics point of view, the most important outcome is the calculation of part demand, which is straightforward, once the product configurations are known. This method works only if the underlying configurations are able to fit with future demand characteristics.

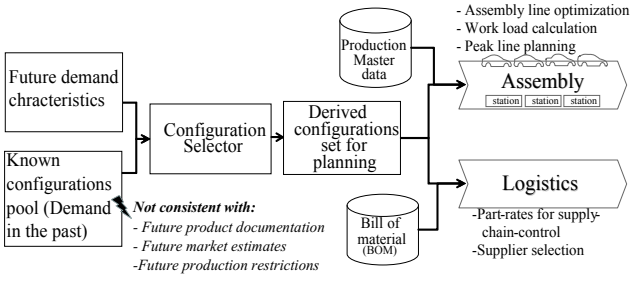


Figure 2: Current Planning: Generation of future customer configuration set through extrapolation of customer demand in past

Variant rich products (e.g automobiles) often receive highly individualized demand and undergo various engineering changes. The regular introduction of new features and short product life cycle make the task of capturing future demand characteristics out of production history a challenging one. In order to facilitate orders/configurations based planning we need to supplement the reference pool of historical production with some customer focused future configurations. To attain planning results of high quality, all the relevant information sources have to be considered, namely the valid list of the product features/attributes, rules for the correct combinations of the attributes, sample of variants produced in the past to capture customers' behaviour, future sales estimates to capture market changes, capacity restrictions and production plans that fix the total number of planned vehicles.

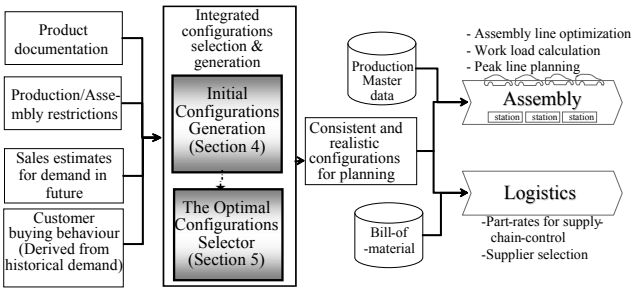


Figure 3: Proposed Framework: Calculation of consistent configuration sets as a foundation for efficient production planning

We propose a different planning set up to generate and select reference configurations for production planning. The main difference between the current (Figure 2) and the proposed methodology (Figure 3) is that current planning is restricted to create production programmes out of known configurations only, while in the proposed approach, we will generate product configurations as and when required, to capture future demand characteristics. This will result in a better match with market estimates and will be consistent with the engineering limitations. Configurations produced in the past can only appear in future planning if they are feasible with latest product documentation. Attributes dependencies in product documentation are maintained formally to support various engineering planning and can be used to automatically check the configurations feasibility. However, historical orders even after failing overall feasibility may contains some important relations among attributes reflecting customer buying habits. For this purpose we will use a data mining approach to identify interesting attribute combinations from the customer point of view, using historical sales data.

The goal of the paper is as follows. Given 1) product documentation 2) market estimates 3) customer behaviour and 4) assembly restrictions, the task is to generate and select valid configurations, which will lead to a production plan. The set of product configurations that are generated is utilized in planning the whole production process (full bill of material, i.e. a car in detail). The optimal configuration selector model (proposed in section 5.2) does not explicitly generate all (or a large number) configurations while it generates a relativity small number of configurations to sequentially build up the desired production plan. The initial configuration generation module (proposed in section 4) is used to provide a starting solution to the optimization model. Although, the optimization based module is able to generate and select configurations, an initial solution from heuristics will give a good starting point. Before describing the development of the solution methodology we list out important data sources and its characteristics.

3.1 Product Documentation

Product documentation is the most important input data for proposed framework and supplies two main sources of data:

1. List of available attributes of the product. For autmobiles, attributes can be power train, Hi-Fi equipment, parking assistant package, etc. Attributes also include labels or user manuals, which may not be crucial for planning but which are required during the creation of an automated framework for production planning such as automated computation of detailed part demand.

2. Rule for feasible combinations of attributes in the configuration. Product configuration can be defined as a list of selected attributes from given set of available attributes. Customer configurations can be produced by combining different attributes all together which are permitted by product documentation. It is important that while combining different attributes, we must fulfil the interdependencies between attributes, so that the feasible product configuration can be generated [Sinz *et al.*, 2003]. For instance, in the USA, some engines required special transmission types, this condition must hold during creating configuration with that particular engine.

Interdependencies among attributes are documented and maintained in the product technical document by a rule system. These rules are basically *propositional Boolean formulas* imposed against each attribute. These formulas are limited to logical operations $\vee$(OR), $\wedge$(AND), $\neg$(NOT). Selection of attributes in a configuration is done through evaluating the respective Boolean formula. Table 1 shows an example of such documentation.

Attribute	Name	Rule	Description
1	Automatic climate control	$(2)\wedge(3\vee4)$	attribute 1 only when attribute 2 is present and either attribute 3 or 4 is present
2	Air condition	TRUE	must be present in every variants
3	Comfort package	$\neg(4)$	attribute 4 should not be present
4	Performance package	$\neg(3)$	attribute 3 should not be present

Table 1: Rule based product document example

The customer order processing is controlled by evaluating the rule's formulae under the variable assignment induced by the customer order and executing suitable action based on whether the formula evaluates to *TRUE* or *FALSE*. [Sinz *et al.*, 2003] have presented a detailed description of one such product documentation; we will use a similar kind of product documentation in this paper. Product documentation describe product in flat structure over attributes (Boolean variables) and capture dependencies through propositional Boolean formulas.

3.2 Future market estimates

Sales and marketing departments continuously study the market behaviour and product positioning. This study enables them to give some demand estimates on key attributes of future products, which any way need to be calculated accurately for various marketing and vendor negotiation purposes. We assume that this information is available to us as an input to capture future market behaviour. We aim to generate configurations sets which represent the given estimates of attributes in the best possible way.

3.3 Assembly/ Production estimates

Product assembly is an important step in customer order fulfilment. Assembly is often restricted to be done on a number of predefined stations with certain work functions at each station. There are some limitations on the capacity and workload at each station. Due to these restrictions, order fulfilment can only be achieved for configurations that satisfy these restrictions. From the aggregate production plan, the total number of planned vehicles can be estimated and the final production plan is generated with the estimated number.

3.4 Customer behaviour

Customer buying trends are extracted by analyzing the product variants produced in the past. We first check the feasibility of variants that are already produced with respect to new product documentation rules. All feasible configurations will be candidates in the solution space. Nevertheless, configurations which are not feasible due to some engineering changes are analyzed on the level of attributes relations. We use the association rule mining technique to identify customer buying behaviour. All relations derived from the data mining approach are again verified with latest product documentation for its feasibility. Customer demand characteristics are calculated as joint or conditional selection rate of attributes, and these are controlled during the development of final production plan. The computation of customer behaviour trends from historical demand is not discussed in this paper and we assume that this information is already available as an input.

4 Configuration Generation

Configuration which satisfies rules from product documentation can be represented as Boolean vector satisfying a constraint system. We want some number of configurations which satisfy product's technical rules and are consistent with customers demand estimates in some way. e.g. we want N configurations which reflects customer demand estimates as best as possible.

As a first step we would like to generate valid configurations which can be use latter for some optimization problem. Generation of configuration involves finding TRUE or FALSE assignment for each attribute. In this section we will propose a guided search procedure which randomly generate configuration with some attribute selected in guided way and others then supplemented as per attribute selection rates form sales. Finally we solve a satisfiability problem with partial assignment of attributes. The satisfiability problem will result selection of generated configuration with some probability.

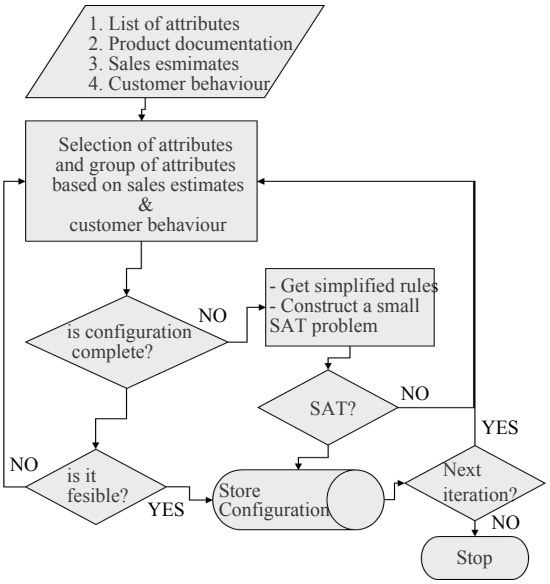


Figure 4: Steps for building up a configurations base

Figure 4 shows a flow diagram of building configurations sequentially. Selection of an attribute in the configuration may exhibit certain characteristics such as mutually exclusivity. For mutually exclusive attributes selection should be done through a multinomial choice. In general, the numbers of possible configurations in mass customization are huge. This motivates us to build a random search procedure to get a representative set of constructible configurations. In principle, first we arrange attributes (or group of attributes) in the decreasing order of their dependency index. The dependency index can be calculated by analysing the selection rule associated with the attribute. The attributes/group which has highest dependency will be selected first and will imply selection (not selection) of other dependent attributes. We will continue to do this process until all attributes assignment is not known. At the end we will check if generated configuration is feasible through evaluating rules with known configuration vector.

As soon as any attribute is selected in the configuration, we check the respective propositional Boolean formula (from product documentation) and select unassigned attributes from Boolean formula such that formula evaluates to *True*. This will keep some level of consistency during the configuration building process. We iterate through this process until a conflict or a steady state where no assignment is possible is reached. If the configuration finds assignment for all attributes, we finally check the overall feasibility of the configuration by evaluation all rules once again. The configuration will be selected with some probability of having a feasible configuration.

If the initial configuration is not able to extract all attributes assignment, we simplify all propositional Boolean formulas with known partial attribute settings and solve a satisfiability problem. Due to assignment of large number of attributes, the number of clauses and literals in satisfiability problems are minimized significantly. If the problem is not satisfiable we reject the configuration and start building a fresh configuration. The configuration generation runs until maximum number of iterations is reached or generated set of configurations is within specified range of attributes estimates.

The randomness in the configuration generation procedure will help in creating diversified configurations that will capture the customer behaviour of individualization in variant rich product. On the drawback side, there will not be any guarantee that the characteristics of generated configurations will improve with number of iterations. This lead us to think about a framework which selects generated configuration such that the target configurations set characteristics can be match as best as possible. At same time, the framework should also be able to generate missing configurations so that characteristics of the final set of configurations can be close to the target one. In the next section we discuss an integrated configurations generation and selection procedure.

5 Integrated configurations generation and selection

Heuristic approach discussed in section 4 does not provide answers for questions like 1) How many constructible configurations will be generated from the configuration generation heuristic? 2) How good the deviation between target and generated set of configurations will be? Although, approach can give a reasonable set of constructible configurations in short period of time this can be used as a starting solution for further optimization process. Based on the quality of starting solution the optimization model can generate missing configurations to complement reference configurations set. To facilitate optimization based approach for generation and selection of configurations we will first transform Boolean propositions from product documentation to a constraints system.

5.1 Transformation of logical rules to linear inequalities

Linear inequalities over Boolean variables are a widely used modelling technique. The main task during transformation of an attribute selection rule into a system of linear constraints is to maintain the logical equivalence of the transformed expressions. The resulting system of constraints must have the same truth table as the original statement. For every attribute we will introduce an the binary decision variable, is denoted by y_i. The connection of these variables to the propositions is defined by the following relations:

$$y_i = \begin{cases} 1 & \text{iff attribute i is TRUE} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Imposition of logical conditions linking the different actions in a model is achieved by expressing these conditions in the form of linear constraints connecting the associated decision variables. Some general transformations are presented in Table 2.

Our approach, in principle, involves identification of precise compound attribute rules of the problem and then processing it with developed equations. Before transformation,

Rule	Description	Constraints
$i \rightarrow j$	i implies j	$y_i - y_j \leq 0$
$i \leftrightarrow j$	i and j must come together	$y_i - y_j = 0$
$i \rightarrow (j \vee ... \vee n)$	if i is true then at least one attributes from j to n, must be true	$y_i - (y_j + ... + y_n) \leq 0$
$i \rightarrow (j \wedge ... \wedge n)$	if i is true then all attributes from j to n (say cardinality p) must be true	$(p)y_i - (y_j + ... + y_n) \leq 0$

Table 2: An excerpt of attribute selection rule transformation table

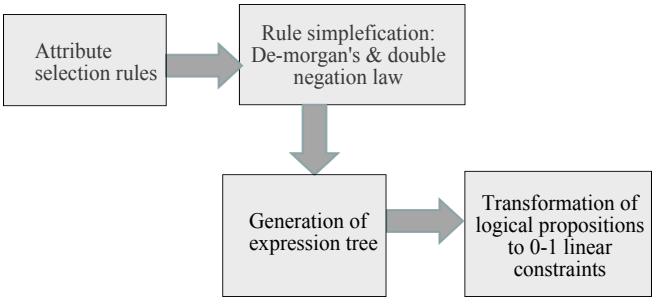


Figure 5: Block diagram for logical rules to inequalities transformation

we simplify Boolean expression through simple Boolean algebra, DeMorgan's Law. The logical rule is represented by a tree graph where attributes are associated with their common operator node. We traverse through the tree and prune the tree in such a way so that standard transformation equation (e.g. Table 2) can be applied. This pruning involves introduction of new auxiliary variables which helps in transformation process.

$$B[y] \leq b \tag{2}$$

As a result of transformation linear constraints sets are created as specified in Eq.2, where B is the constraint matrix contains all constrains originating from product documentation. All product configurations must satisfy Eq.2 in order to be feasible for production and can be a candidate for production plan.

5.2 The optimal configuration selection model

To create production plan based on detailed product configuration we need to list out some number of configurations (say K) in such a way that estimated characteristics of configurations can be match as best as possible. For example, let us assume that an automotive contains 1000 of attributes and we want to select 3000 configurations generated from available attributes. Our task will be to answer, does attribute i belong to the configuration j finally selected? This example will leads to s3 millions (a large number) of 0-1 type decision variables. We do know something about the portion of attribute (i's) in the final configurations (demand estimates from sales, customer behaviour etc.). So the objective function will be to minimize the positive deviation between selected and estimated values. General structure of above problem is defined over combinatorial optimization, with a very large number of variables that is quite difficult to solve.

Another possibility of formulation for given problem is to list all possible configurations with the given number of attributes. This runs into the hundreds of millions! Much larger than the previous formulation. We can define 0-1 variable over each configuration on whether it is selected or not. These feasible configurations has to be implicitly represented, i.e. not possible to list all of them explicitly. Surprisingly, this way of thinking is still useful. In this section we will develop an optimization model based on Lagrangian approach using column generation.

The Master Problem:
Let: i be i^{th} attribute, $i \subseteq \{1...I\}$, where I is the number of attributes
j be j^{th} configuration, $j \subseteq \{1...J\}$, where J is the number of unique configurations

Data
K = the number of configurations planned
D_i= Demand estimate for attribute i
C_i= Demand mismatch cost associated with attribute i
λ= Lagrange multiplier

$$A_{i,j} = \begin{cases} 1 & \text{if } i^{th} \text{ attribute is present in } j^{th} \text{ configuration} \\ 0 & \text{otherwise} \end{cases}$$

Decision variables:

$$X_j = \begin{cases} 1 & \text{if configuration } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

Z_i= Deviation between given demand estimate and calculated frequency for attribute i

Objective Function:

$$\textbf{P = Minimize } \sum_i C_i * Z_i + \lambda(\sum_j X_j - K) \tag{3}$$

Subject to

$$Z_i \geq \sum_j A_{ij} X_j - D_i \ldots \forall i \tag{4}$$

$$Z_i \geq D_i - \sum_j A_{ij} X_j \ldots \forall i \tag{5}$$

$$X_j = 0 \text{ or } 1 \tag{6}$$

The first sum of objective function in Eq. 3 tries to minimize mismatch cost of the positive deviation from desired demand estimates of the attributes. Second part of objective function ensures that selected number of configurations are closed to K, the desired number for planning. While, constraint Eq. 4 Eq. 5 ensure the feasible configuration set close

to attribute demand estimates. Each attribute is associated with a demand mismatch cost. We fixed X_j to 0-1 type to support higher degree of individualization in the generated plan.

The above model is too large to solve by explicitly generating large number of configurations. A solution for such a large scale optimization can be found using column generation approach [Ben Amor and Valrio de Carvalho, 2005]. We can start with a possible set of X_j variables (may be more than K, generated from Section 4 or derived from history) Solving LP relaxation of the above problem to decide which of those X_j's are 1, and then try generate a new configuration which can improve the objective function value.

5.3 Implementation of the model

To start solving the model presented in Section 5.2 we use a fixed value of λ (this is because anyway we want approximately K configurations that are representative of the demand). Now we can solve the master problem with initial sets of X_j's. The question would be how to know if the current selection of configuration is good. For this, we will compute dual variable corresponding to constraints 4-5, with these values a sub problem is set up. The sub problem is basically a generation of new configuration for A_{ij} matrix which can be formulated as follows:

The Sub-Problem:
Data: W_i= Dual variable from LP relaxed of the optimal configuration selection model (5.2), associated with constraints 4-5 (note that for each i, one of them will be non-zero)
B= Set of constraints derived from product document (see Section 5.1)
Decision Variable

$$[y_j]_i = \begin{cases} 1 & \text{if } i^{th} \text{ attribute is present in new configuration} \\ 0 & \text{otherwise} \end{cases}$$

$[y_j]_i$= new configuration for j^{th} column of configuration matrix $A_{i,j}$

Objective:

$$\textbf{Maximize} \sum_i W_i * y_i \tag{7}$$

subject to:

$$B[y] \leq b \tag{8}$$

(i.e. y is a feasible configuration)

$$y_i = 0 \text{ or } 1 \tag{9}$$

The sub-problem is generate a possible new configuration j. If this new configuration j satisfies Eq. 10 the configuration j enter the pool.

$$\sum_i A_{ij} + \lambda - \sum_i W_i * y_i < 0 \tag{10}$$

Dual costs are recomputed by solving the master problem (Section 5.2) and the process terminates when no more configurations are found to be worth taking in. We use IBM ILOG Cplex engine to solve the sub-problem. The master and the dual problem may have to be solved multiple times before terminating criteria satisfies.

6 First evaluation results

In this section we discuss typical computation parameters and associated numbers with input data and decision variables. We tested our methods mainly on automotive data, for configuration based planning the granularity of the computation is plant/model series/body style (e.g. Bremen, C-class, Sedan). All input data and estimates are available or derived to same granularity. Total number of attributes are in the range of 500-1000. Typically selection rates of 100-200 key attributes are estimated from sales. Some attributes are related to production such as plant where production takes place, regularity laws, dependencies structure because of technical reason.

We target to generate production plan for weekly or monthly time frame and requite to simulate some thousand of configurations, typically 3-10 thousands of configurations in one computation. The generation of production plan with thousands of configuration need to be done by ensuring maximum correlations with given demand characteristics (e.g. sales estimates). The typical use of calculated configurations is to derivation of part demand or estimation of medium term workforce in assembly operations.

On an example with 130 attributes (for which attached selection rates are given), 900 total attributes and selection of about 3000 configurations, resulting problem has 10,000 variable and 15,000 constraints. The match between the target and achieved frequency of attributes in generated configuration set is defined by the ratio of target and gain frequency of attributes. If this ratio is equal to one, it is desirable.

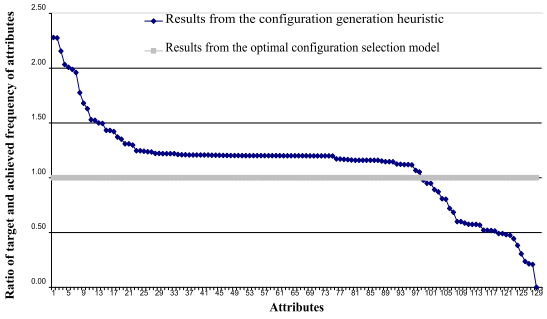


Figure 6: Attribute frequency match between target demand and gain rate in generated configuration set

Figure 6 provides comparison between results obtained after configuration generation heuristic (Section 4) and optimization based procedure (Section 5). The grey (light) line represents the best solution of the optimization model, which is very close to 1. The result from the configuration generation heuristic is plotted in a decreasing order of deviation from the target rate (in blue). We can see that some of the attributes are more than required and some less, but many are quite close to the desired target. This can perhaps be improved further in the satisfiability section of the algorithm, which allows different heuristic ways of completing orders.

In general, the heuristic provides a starting solution for our optimization model and helps in speeding up in the process of generating a configuration set for planning. The attribute selection rate obtained in generated configuration set matches reasonably with the target attribute demand.

7 Conclusion

In this paper we presented a production planning framework based on fully specified products which guarantees consistency among different planning tasks. The mathematical model that has been developed is capable of considering heterogeneous information generated by different planning departments. In this framework we are able to consider up-to-date product documentation at an early stage of program planning. The problem of find a valid set of configurations is formulated as an optimization problem by translating all logical conditions from the product document to algebraic inequalities. This transformation enables us to use the optimization framework effectively. The number of constraints generated during this transformation can be further reduced by simplifying the product rule system or through pattern identification in the product documentation.

The large variety in products implies that the construction of the set of customer-focused configurations is a large scale optimization problem. Our proposed column generation approach can be useful to get a good solution for this problem. The configuration generation heuristic is based on the guided search procedure that can be enhanced further to gain better speed and improve the result quality. Some other parameters like restrictions at the part level and assembly operation level are subjects of future research.

References

[Aldanondo and Vareilles, 2008] Michel Aldanondo and Elise Vareilles. Configuration for mass customization: how to extend product configuration towards requirements and process configuration. *Journal of Intelligent Manufacturing*, 19(5):521–535, 2008.

[Astesana *et al.*, 2010] Jean-Marc Astesana, Laurent Cosserat, and Helene Fargier. Constraint-based vehicle configuration: A case study. In *Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '10, pages 68–75, Washington, DC, USA, 2010. IEEE Computer Society.

[Ben Amor and Valrio de Carvalho, 2005] Hatem Ben Amor and Jos Valrio de Carvalho. Cutting stock problems. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 131–161. Springer US, 2005.

[Brabazon and MacCarthy, 2004] Philip G. Brabazon and Bart L. MacCarthy. Virtual-build-to-order as a mass customization order fulfilment model. *Concurrent Engineering: Research and Aplicxations*, 12(2):155–165, 2004.

[Harrison *et al.*, 2004] Terry P. Harrison, Hau L. Lee, John J. Neale, S. Venkatesh, and Jayashankar M. Swaminathan. Managing product variety through postponement: Concept and applications. In Frederick S. Hillier, editor, *The Practice of Supply Chain Management: Where Theory and Application Converge*, volume 62 of *International Series in Operations Research and Management Science*, pages 139–155. Springer US, 2004.

[Hayler, 1999] Christian Hayler. *Ein regelbasiertes System zur Generierung von Orders für Lagerfahrzeuge-Fallstudie bei einem deutschen Automobilhersteller*. PhD thesis, Universität Jena, 1999.

[Kaiser and Küchlin, 2001] A. Kaiser and W. Küchlin. Automotive product documentation. In *Proceedings of the 14th International IEA/AIE Conference, LNCS*, pages 465–475. SpringerVerlag, 2001.

[Olsen and Saetre, 1997] K. A. Olsen and P. Saetre. Managing product variability by virtual products. *International Journal of Production Research*, 35(8):2093–2108, 1997.

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

[Sinz *et al.*, 2003] C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1):75–97, January 2003. Special issue on configuration.

[Srinivasan and Swaminathan, 1997] R. Srinivasan and J. M. Swaminathan. Managing configurable products in the computer industry: Planning and coordination issues. volume 22, pages 33–43. Sadhna:Academy Proceedings in Engineering Sciences, February 1997.

[Stautner, 2001] Ulrich Stautner. *Kundenorientierte Lagerfertigung im Automobilvertrieb: ergnzende Ansätze zum Supply-chain-Management*. PhD thesis, Universität Göttingen, 2001.

[Swaminathan and Tayur, 1998] Jayashankar M. Swaminathan and Sridhar R. Tayur. Managing broader product lines through delayed differentiation using vanilla boxes. *Manage. Sci.*, 44:161–172, December 1998.

[Trentin *et al.*, 2011] Alessio Trentin, Elisa Perin, and Cipriano Forza. Overcoming the customization-responsiveness squeeze by using product configurators: Beyond anecdotal evidence. *Computers in Industry*, 62(3):260–268, 2011.

Choice Navigation: Towards a Methodology for Performance Assessment

Simon Haahr Storbjerg, Kjeld Nielsen and Thomas Ditlev Brunoe
Department of Mechanical and Manufacturing Engineering, Aalborg University, Denmark
shs@m-tech.aau.dk

Abstract

Based on the increased demand for product customization and the intensified competition, manufacturing companies are today more than ever required to deliver product variants in an efficient manner. Research on mass customization has, up until now, primarily focused on clarifying the organizational capabilities defining successful mass customizers. Choice navigation is identified as one of the three fundamental capabilities. The process of building this capability does not occur as a discrete event, it is a change process. Based on literature review and analysis, this paper addresses the change process in relation to implementation of the choice navigation capabilities. A framework for performance assessment, supporting implementing of the choice navigation capabilities, is forwarded.

1 Introduction

A broadly recognized trend of today's markets is the demand for customized products and services meeting the individual customer's needs. Simultaneously today's manufacturers are faced with demands for delivering products faster and cheaper. These market trends happen in concurrence with the increased saturation and globalization of markets. Consequently, today's manufacturers are on top of the demand for customization, also faced with increasing demands for operating in an effective & efficient manner.

Perfectly suited to this challenge, mass customization arose as a concept and an operations strategy in the late 80's, combining the ability to deliver products that meet the individual customers' needs, as well as having an efficiency similar to mass production [Davis, 1989]. Since then, research has focused on clarifying the fundamental, or defining, characteristics of the firms that successfully adopts the mass customization strategy. This has led to the introduction of three fundamental dimensions in enabling the mass customization ability. The three dimensions are by [Salvador et al., 2009] framed as the three fundamental mass customization capabilities; Solution space development, robust process design and choice navigation.

This paper focuses on the process of building the choice navigation capability. This capability, or rather set of capabilities, refers to the ability to support customers in the process of selecting the solution or variant that fulfils the customer requirements out of a pre-defined solution space, and maximizes the customer value. Several researches and practitioners in the industry have adopted the three fundamental capabilities, and continued this line of research, defining and developing a more comprehensive understanding of what characterises and constitutes a successful mass customizer [Fogliatto et al., 2012; Lyons et al., 2012; Piller & Tseng, 2010; Walcher & Piller, 2011]. However, recent studies report that experience in industry adopting and building these capabilities, is for many companies an unsuccessful quest, leading to in worst cases company closures [Piller et al., 2012b]. Based on this knowledge, we argue that the industry lacks more detailed and comprehensive guidance, on how to undertake the transition from conventional approaches at manufacturing, to mass customization as a manufacturing strategy. Research on mass customization has also lately increasingly focused on the "how to" of mass customization, in order to provide improved guidance for companies in the organisational transition, when following a mass customization strategy, e.g. [Partanen & Haapasalo, 2004; Pollard et al., 2011].

The same situation holds true when focusing on choice navigation. Significant amount of research and valuable knowledge have been generated on what choice navigation is about, including how to develop product configuration systems. However, the topics of how to support the transition towards MC, and additionally the process of building the choice navigation capabilities, have thus so far only been scarcely addressed.

An alternate method of supporting organisational change, which is often addressed in other streams of literature, is the use of performance management. In relation to this, Nielsen, Brunø and Jørgensen [Nielsen et al., 2012] have introduced an overview of metrics and a framework for measuring a company's performance as a mass customizer. However, as the metrics only focus on solution space assessment or mass customization in general, no guidance is given in regard to choice navigation.

The purpose of this paper is based on this shortcoming in the existing literature on mass customization to answer the following research questions:

How can performance assessment support the implementation of the choice navigation capabilities? What performance assessment methodologies are appropriate?

In order to answer this, the choice navigation capability is

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

further detailed in the following section by the use of central literature In section 3, a model is introduced describing the dimensions along which performance assessment is relevant in the context of choice navigation. Based on this model, relevant performance assessment methodologies are based on the literature review introduced in section 4. In conclusion, the results of the literature review are discussed, and direction for potential further research is given.

2 Choice Navigation - What is it about?

What performance assessment methodology is appropriate depends on the object or artefact of measurement, as this defines what is relevant to measure, and how measurement can be done. As the fundamental capabilities of mass customization are defined at a rather abstract level, it is challenging to relate this to specific activities, or activity-areas, in a firm. Based on the aforementioned premise, the principal questions are: What is choice navigation really about? What does the choice navigation capability mean in an industrial context? Which activities, systems and human competencies does this abstract and high level capability refer to?

The choice navigation capability is by [Salvador et al., 2009] defined as, the capability of "supporting customers in identifying their own solutions, while minimizing complexity and the burden of choice". By this definition it is revealed that, the concept of the choice navigation capability, builds on assuming a causal relation between the efforts required of the customer to identify the solution, and the customer satisfaction. Consequently when customers e.g. are exposed to an assortment of too many choices, the cognitive cost of evaluation outweighs the value of increased variety [Huffman & Kahn, 1998; Piller et al., 2012a]. Based on this knowledge, companies are required to simplify the navigation of their product assortment.

It could seem as if MC-scholars are more or less in agreement on the underlying phenomena of choice navigation. However, if the literature on mass customization and choice navigation is reviewed, it is revealed that the conception of the choice navigation capability varies.

Some authors, e.g [Da Silveira et al., 2001; Fogliatto et al., 2012] describes choice navigation as a customer manufacturer communication, involving the transfer of knowledge from manufacturer to customer, and vice versa. Hence a knowledge transferring process done by so-called agents of information transfer, which in this connection are described as the manufacturer and its customers. In contrary to this, other authors, e.g. [Franke & Piller, 2003; Heiskala et al., 2010; Mortensen et al., 2008; Trentin et al., 2013] describe choice navigation, as a configuration system involving the use of dedicated IT support, in the form of a product configurator, also referred to as choice board, or customer design system.

Investigating the underlying view of the choice navigation capability in these cases, it is evident that in both [Da Silveira et al., 2001; Fogliatto et al., 2012] the choice navigation capability is described as primarily relating to the agents of information transfer, whereas the view on choice navigation in the perspective of [Heiskala et al., 2010] primarily relates to the configuration system, its features, user interface layout and ability to configure a variety of products as well as undertake data migration.

Instead of arguing for or against these different views, the choice navigation capability has more recently by e.g. [Forza & Salvador, 2007] also been described from a more holistic perspective. Building on this, the implementation of the choice navigation capability is more than just implementing a configuration system, it is about managing organizational change, which involves both changes in systems and people. Following this, we suggest that this process should be viewed from a socio-technical perspective [Trist, 1981].

2.1 Choice Navigation from a Socio-Technical System Perspective

Viewing this concept from a socio-technical point of view, it is implied that a company's capability to perform choice navigation does not rely entirely on the technical systems, but to some extent also on the people using the system, whether internal sales people or external customers.

Based on the above, we argue that choice navigation as depicted on Figure 1, consists of both social assets, such as behaviour, routines and skills of e.g. sales personnel, as well as technical assets such as information systems, tools etc. Based on this, we argue that the choice navigation capability is to be viewed as a higher level abstract capability, which is constituted by a set of more concrete capabilities.

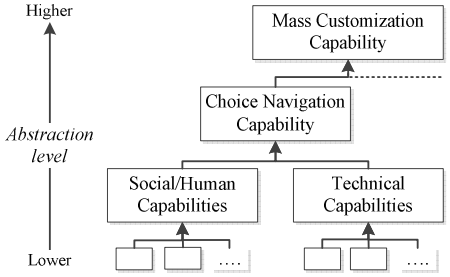


Figure 1 Choice navigation as a socio-technical system capability with multiple abstraction levels.

Another argument for taking a more holistic and socio-technical system perspective on the choice navigation capability, is found in the following definition of capabilities, which both encompasses human assets, and technical assets. According to [Boer et al., 2001], capabilities can be described as "Integrated stocks of resources that are accumulated over time through learning, or established through deliberate decisions. These stocks of resources include internalised behaviours, technical skills, organisational routines, and corporate assets such as information systems, databases, libraries, tools, and handbooks".

3 Transition Towards Choice Navigation

Mass customization calls for a transformed company [Boynton et al., 1993]. As highlighted by [Salvador et al., 2009], this transformation is not something that can be

realized in a single event, it is an on-going or continuous improvement activity.
The purpose of this paper is to clarify performance assessment methodologies, that can give valuable feedback on the implementation of the choice navigation capabilities, so that corrective actions can be taken.
Based on the viewpoint that the choice navigation capability is comprised of both social and technical capabilities, key questions in relation to this are: *How to understand and model the process of building the choice navigation capabilities? Which performance constructs can be identified, i.e. along which dimensions can performance of this socio-technical configuration system be described?*
In relation to the first question [Boer et al., 2001] has introduced the model depicted in Figure 2, which describes the central constructs in the process of building capabilities for continuous innovation.

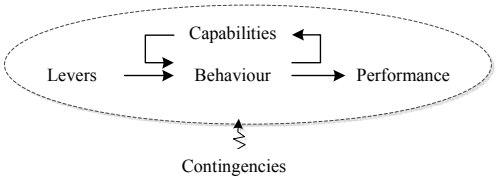


Figure 2 CIMA behavioural model by [Boer et al., 2001].

As the model in Figure 2 links elements such as capabilities, performance and levers, we have chosen to take point of departure in this, in modelling of the central elements involved in implementing the choice navigation capabilities. The outcome, which is depicted at Figure 3, shows how the choice navigation process, which consists of interplay between behaviour of the technical system and the social system, determines the choice navigation performance. Furthermore, the choice navigation process is affecting the choice navigation capabilities, by e.g. development of routines based on repeated behaviour. The choice navigation process is in turn affected by the capabilities of the company, and the levers brought in use, e.g. IT systems, etc. Finally the levers utilized are based on feedback or control information from the performance of the process.

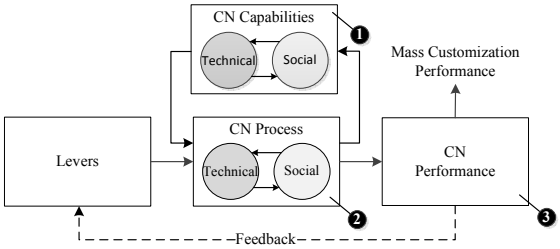


Figure 3 Behavioural model of the socio-technical CN system, outlining the three dimensions of performance assessment. Model is based on modifications to model of [Boer et al., 2001].

Based on the constructs of this process in building the choice navigation capabilities, three dimensions have, as depicted at Figure 3, been identified potential in describing the performance of this process:
1) The degree to which the capabilities have been built
2) The choice navigation process performance
3) The output performance of the choice navigation process

In addition to these three performance dimensions, it is also relevant to describe the performance of the mass customization process. This is however not included as an additional dimension, as it is believed to be hard to distinguish between the performance of choice navigation, and the performance of the mass customization process.
According to the three aforementioned dimensions, as well as literature review, relevant performance assessment methodologies are introduced in the subsequent section.

4 Performance Assessment Methodologies

It has for long been recognized that performance assessment has an important role to play in the efficient and effective management of organizations [Kennerley & Neely, 2003]. This topic has, as reckognized by among others [Folan & Browne, 2005], also gained focus in an ever-increasing number of academic fields.
The research on performance assessment was initiated in management accounting in the beginning of the 20th century, and later gained a broader role into non-financial disciplines, such as operations management, marketing, and human resource management [Chenhall & Langfield-Smith, 2007]. Organisational performance is as highlighted by [Cameron, 1986] among others, by no means a simple phenomenon; rather, it is a complex and multidimensional concept. There are several purposes of conducting performance assessment, [Melnyk et al., 2004] highlights one which quite accurate defines the purpose of performance assessment in this context;
"closed-loop deployment of organizational strategies, allowing relevant information to feed back to the appropriate points facilitating decision and control processes".
Assessment of organisational performance, in order to provide control information, has split into two main streams in literature; one stream focusing on metrics, performance measures, performance measurement systems, and approaches to performance management, e.g. [Folan & Browne, 2005; Melnyk et al., 2004; Neely et al., 2005]. The other stream of literature, which is dominatantly within quality management literature, focuses more on the use of capability maturity frameworks, in the assessment of organisational capabilities, e.g. [Maier et al., 2012].
Despite different approaches and focus, the two streams of literature both provide methodologies for feedback, recommendations and control information enabling assessment of an improvement effort. In order to clarify what performance assessment methodologies are appropriate, central contributions within each of these streams are reviewed in the following, and reference is given to the three performance dimensions identified in above.
The performance measurement methodologies are assessed agains three criterias:
1) *What is measured?* Do the methodology encompass performance assessment by quantitative performance measures or assessment of organizational capabilities?

2) *Non domain-specific?* Are the methodology non specific for a particual domain, i.e. are the methodology more generally applicable.
3) *Operationalizable?* Are the methodology operationalizable, i.e. not only conceptual.

Only the performance measurement methodologies meeting the three requirements are introduced in the following.

4.1 Performance Measurement and Management

Performance measurement has its roots in early accounting systems, the first financial ratios and budgetary control procedures was developed in DuPont and General Motors during the early 1900s [Neely et al., 2005]. Since then the demands from managers, to assess the effectiveness and efficiency of specific areas, have resulted in a proliferation of approaches to performance measurement [Chenhall & Langfield-Smith, 2007]. Today, basically all areas of an organisation are in the scope of performance measurement and management, each with distinct perspectives and purposes.

The research on performance measurement can according to [Folan & Browne, 2005], be said to give recommendations on four different levels or dimensions. Recommendations for:

1) Individual performance measures
2) Structural frameworks (set of performance measures)
3) Procedural frameworks (process of building performance measures systems)
4) Performance measurement systems (the integration of the above)

The term performance framework refers, as stated in [Folan & Browne, 2005], to the active employment of particular sets of recommendations. What is in common for most of the performance measurement frameworks and systems are, that the performance measurement boundaries, dimensions and relations in between the measures are given.

Rather than giving an extensive review on literature on performance measurement and management, the objective of this paper is more to clarify performance assessment methodologies relevant for choice navigation.

Based on this focus, the literature review concentrates on performance measurement systems and structural frameworks. Literature on individual metrics and literature on procedural frameworks are thus omitted. For a review of individual performance measures we refer to [Chenhall & Langfield-Smith, 2007] . Similarly, for a more extensive review of the available performance measurement frameworks we refer to [Folan & Browne, 2005; Neely et al., 2005; Pun & White, 2005].

The performance measurement methodologies found relevant based on the criterias listed in the following. For each method, it is in brackets indicated, which of the performance assessment dimensions, depicted at Figure 3, the metholody is supporting.

AMBITE performance cube [2,3]

The structural performance framework introduced by [Bradley, 1996] is specifically designed to suit a so called Business Process Reengineering process. The framework consists of three dimensions:

- Business processes: customer order fulfilment, vendor supply, engineering, manufacturing, etc.
- Competitive priorities: time, cost, quality, flexibility, environment
- Order-delivery type: make-to-stock, assemble-to-order make-to-order, engineer-to-order.

With regard to these three dimensions, combinations of different strategic performance indicators (SPI's) can be generated. Each of these strategic performance indicators can be broken down into lower level indicators. This breakdown is done context specific, and the performance indicators are thus customised to the context of application. In addition to the structural framework [Bradley, 1996] also introduce a procedural framework for PM system design. This describes how to link the performance indicators with the company's strategy statement and business processes.

Balanced Score Card (BSC) [2,3]

One of the most recognized and broadly applied performance systems or frameworks is the BSC, which was developed by [Kaplan & Norton, 1992]. The BSC approach gives a holistic view of the organization by simultaneously looking at four different perspectives on performance; (1) Financial, (2) internal business, (3) customer, (4) innovation and learning. BSC is based on this a good example of a performance assessment system that employs a balanced set of financial and non-financial measures. The BSC approach is based on the principle that a performance system should provide managers with sufficient information to address the following questions:

- How do we look to our shareholders (financial perspective)?
- What must we excel at (internal business perspective)?
- How do our customers see us (customer perspective)?
- How can we continue to improve and create value (innovation and learning perspective)?

The performance measures to be utilized in the BSC system is initially to be formulated during the system development process, according to the BSC system design methodology.. Based on this, no performance measures are explicitly pre-defined by the approach.

Comparative Business Scorecard [2,3]

With point of departure in the balanced scorecard, [Kanji, 1998] introduced the Comparative Business Scorecard. This framework is based on adaption of TQM principles to monitor progress and performance toward towards excellence. To enable this the performance measures focuses on the drivers of success; delight the stakeholders, ensure stakeholder value, process excellence and organisational learning.

As noted in [Kanji, 1998] this framework is merely an attempt to go a step further and extend the understanding of the four BSC perspectives. The framework is in methodology and structure, thus not radically different than the BSC.

General Motors Integrated Performance Measurement System [2,3]
This integrated performance measurement system is an outcome of significant investments within General Motors in the early 90's, in the design of a performance measurement and feedback system, consisting of 62 measures [Gregory, 1993]. The framework is, in order to provide valuable input in a complex organisation, designed to be applied at various organisational levels, with specific measures for each level. The measures can generally be split in measures of results, e.g. quality and responsiveness, and measures of the process of strategy implementation. The measures ensures that employees retain their focus on continuous improvement through teamwork in the key business activities.

Integrated Performance Measurement Framework [2,3]
Similarily to the approach of General Motors, the Integrated Performance Measurement System of [Medori & Steeple, 2000], encompasses multiple measures. The structural performance framework is composed of five sub-systems each with distinct purposes of performance measurement, and each with different performance measures. The five sub-systems of the performance framework interact and co-ordinate in a controlled fashion. The integrated performance framework does not include any procedural elements, besides a set of principles that should be considered alongside the framework.

Performane Prism [1,2,3]
The Performance Prism framework introduced by [Neely et al., 2002] offers a new approach to measuring organisational performance in that it integrates strategy, capabilities and performance measures. The framework is built upon the argument that one of the greatest fallacies of measurement design is that performance measures should be derived from strategies.
The performance framework includes five inter-related and weighted aspects;

1) Stakeholder satisfaction; who are the organization's key stakeholders and what do they want and need?
2) Stakeholder contribution; what contributions does the organization require from its stakeholders?
3) Strategies; what strategies does the organization have to put in place to satisfy the wants and needs of these key stakeholders?
4) Processes; what critical processes does the organization need to operate and enhance these strategies?
5) Capabilities; what capabilities does the organization need to operate and enhance these processes?

To each of the aspects of the framework specific performance measures are given, accompanied by their results, trends, targets, standards, initiatives and action plans. These data-sets are included in scorecards to facilitate the performance management. The measurements are furthermore connected with each other through sets of hypothetical relationships called "success map". Together the five viewpoints provide a comprehensive and integrated framework for managing organisational performance.

Results and Determinants Matrix [2,3]
The performance measurement framework introduced by [Fitzgerald et al., 1991] is especially developed for the services businesses. The framework employs a distinction between measures of results, and measures of the determinants of the results. The frame work involves several measures, e.g. competitiveness, liquidity, capital structure and market ratios, that according to the author do not vary across the three generic service types, which is identified.

Strategic Measurement Analysis and Reporting Technique (SMART) [2,3]
The Strategic Measurement Analysis and Reporting Technique (SMART) system, also known as the Performance Pyramid, is designed by [Lynch & Cross, 1992]. The system is designed with the intent of creating a management control system of performance indicators, that can assist in defining and sustaining organisational success. The framework employs a hierarchical view of business performance measurement, in the sense that it is modelled as a pyramid with four hierarchical levels of objectives and measures. The SMART system includes a 10 step procedural framework describing the performance assessment process.

Structural Performance measurement matrix [2,3]
[Keegan et al., 1989] have proposed a structural performance measurement framework which seeks to integrate different dimensions of performance. The framework is modelled as a 2x2 performance measurement matrix, that categorises performance measures based on two dimensions; financial versus non-financial and internal versus external.

In addition to the performance measurement systems described in above, a number of more conceptual performance measurement systems have been identified; Dynamic Performance Measurement Systems (DPMS) Integrated Performance Measurement Systems (IPMS), Framework for multinational companies, and the ICAS performance measurement framework. Furthermore, a number of more procedural focused performance measurement systems or frameworks have been identified, for an overview of these we refer to [Browne et al., 1988].

4.2 Capability Assessment Methodologies

The assessment of organisational capabilities, is another promising way of providing feedback and control information in process improvement initiatives. The purposes or drivers for adopting a capability based approach to performance assessment are however, as highlighted by [Maier et al., 2012], other than process improvement. Some might adopt capability assessment based on imposed conformance requirements. In other cases customers may explicitly require compliance with certain frameworks, or the competition on the market place may implicitly require compliance.
Capability assessment frameworks are generally designed to assess the maturity of either the entire organization, or a

selected domain, e.g. process or functional area. The capability assessment is typically conducted by appraisal of the activities done, against a predefined set of criteria's, which most often is gathered in a framework. Process improvement is a central Total Quality Management (TQM) concept, and much of the research on capability maturity assessment, has been done within quality management. The use of capability maturity assessment frameworks has since the concept of measuring maturity was introduced in the early 90's proliferated across a multitude of domains.

The work on capability framework can generally be split up into capability maturity models, and capability grids, which according to [Maier et al., 2012] can be distinguished on three aspects; work orientation, mode of assessment and intent.

As with the performance measurement frameworks, the aim of this paper is not to conduct an extensive review, due to this only the grids and maturity models that are identified as relevant in this context, are addressed in the following. For a more comprehensive review of capability assessment frameworks we refer to [Maier et al., 2012].

Based on an extensive literature search [Maier et al., 2012] have identified 61 maturity grids. Before conducting the review, the number of methodologies for review have been narrowed down to 24 based on requirement to among other things a grid-based approach. Utilizing the criterias from section four in the review of these grids, five grids have been identified relevant.

Similarly [Kohlegger et al., 2009] review based on extensive literature search, and preliminary filtering, 5 maturity models. If the three criterias listed introductory in section 4 are utilized in evaluation, only the CMM model is found relevant.

The capability assessment metholodgies found relevant is described in the following. It is for each indicated in brackets which of the performance assessment dimensions depicted at Figure 3 the methodology is supporting.

Capability maturity models (CMM) [1]

The Capability Maturity Models (CMM) was first developed at the Software Engineering Institute (SEI) at Carnegie Mellon University [Paulk et al., 1993]. Where the focus of the first CMM models was to support assessment software development within a number of sub-processes, an integrated capability maturity model (CMMI) has later been introduced [Chrissis et al., 2003].

The integrated model consists of 22 process areas, and supports product development in general. The capability maturity model works as a multi-level maturity ranking process, where a number of important areas, relative to an organisations' performance, have been clarified. For each of these areas a number maturity levels has been defined, each with distinct capabilities, i.e. practices, methods, skills, tools etc. By auditing the practices done in a company, the capabilities and maturity levels can be identified. Due to this, progressively greater levels of performance are reflected, as an organisation matures in general or within specific areas.

Communciation Grid [1]

Based on the stand that effective communication is key to avoid problems within engineering design, the communication maturity grid has been developed by [Maier et al., 2006]. The purpose of this framework is to assess the maturity of the communication of the engineering design activities. The grid measures the maturity within 5 process areas against four generic maturity levels.

Design Process Audit Grid [1]

A good design is key for company success. Based on this [Moultrie et al., 2007] has developed the design process audit grid. The grid is developed to assess the maturity of the design processes within SME's. Based on 24 process areas the activities in design from requirements capture to introduction in manufacturing are asessed against four maturity levels.

Innovation Audit Maturity Grid [1]

The innovation audit maturity which is introduced by [Chiesa et al., 1996], focuses on the product development processes through which innovation and innovation management is performed. The grid consists of 8 process areas each with 2-4 sub-questions. The audit methodology uses a two level approach a rapid assessment and an in-depth audit.

Product and Cycle time Excellence Maturity Grid [1]

The purpose of the Product and Cycle time Excellence (PACE) maturity grid is to assess and improve the progression of the new product development process [McGrath & Akiyama, 1996]. The PACE maturity grid encompasses 10 process areas related to product development, and measures against four levels of maturity.

R&D Effectiveness Maturity Grid [1]

The maturity grid for measuring R&D Effectiveness is developed by [Szakonyi, 1994] based on several decades of experience and work with a number of companies. The framework measures 10 processes related to R&D.

5 Conlusion & Discussion

There seems to be general agreement between the industry and academia that the competition on the market place displays a trend of higher price competition combined with the demand for customization. The requirement of companies to meet the individual customers' demand at a reasonable price continues to characterize a central challenge for industrial manufacturers today. Based on this, successfully managing the radical organizational change that following it requires to follow a mass customization strategy, is still an important topic. The purpose of this paper is to support clarification of a methodology for assessing the performance of the choice navigation process. The aim of the research is to enable an improved management of the organizational change in the process of building the choice navigation capabilities.

According to the conducted literature review and analysis, a variety of methods for giving feedback and control information on performance have been clarified. In

answering if any methods are appropriate for giving relevant feedback information to the process of implementing the choice navigation capabilities the first step is to review and discuss the available methods at a typological level.

Two types of performance assessment methodologies are identified from existing literature on quality management and process improvement; 1) performance measurement systems and 2) capability maturity assessment frameworks.

Use of metrics in performance measurement systems enable the provision of information on the output performance of the choice navigation process is. As highlighted by [Neely et al., 2005] this enables that the efficiency and effectiveness of the process can be quantified.

Another type of input is given if the capabilities in relation to the choice navigation process are assessed. As noted by Maier this type of assessment enables that the maturity of the process, understood as what collective assets, e.g. skills, routines, tools, systems etc. have been built around the process can be evaluated.

We consider both types of performance assessment as highly relevant in giving feedback information to the process of implementing the choice navigation capabilities. Based on this we suggest that the discussion is more centralized on how to actually combine these, than on which is most beneficial. As a first step in establishing a combined and customized methodology for performance assessment, the existing methodologies need to be assessed. For this purpose the focal paper contributes to existing literature on mass customization with a socio technical system model describing which constructs are relevant in the performance assessment. With the use of this model, the existing literature on performance assessment is reviewed and classified. The research thus enables that a performance assessment metholodogy supporting the building of choice navigation capabilities can be proposed based on further research.

References

[Boer, H., et al. 2001] Harry Boer, Sarah Caffyn, Mariano Corso, Paul Coughlan, José Gieskes, Mats Magnusson, Sara Pavesi and Stefano Ronchi. Knowledge and continuous innovation: The CIMA methodology. *International Journal of Operations & Production Management*, *21*(4), 490-504. 2001

[Boynton, A. C., et al. 1993] Andrew C. Boynton, Bart Victor and II Pine. New competitive strategies: Challenges to organizations and information technology. *IBM Systems Journal*, *32*(1), 40-64. 1993

[Bradley, P. 1996] P. Bradley. A performance measurement approach to the re-engineering of manufacturing enterprises. *CIMRU, NUI Galway*, 1996

[Browne, J., et al. 1988] Jimmie Browne, John Harhen and James Shivnan. *Production management systems: A CIM perspective* Addison-Wesley UK.1988

[Cameron, K. S. 1986] Kim S. Cameron. Effectiveness as paradox: Consensus and conflict in conceptions of organizational effectiveness. *Management Science*, *32*(5), 539-553. 1986

[Chenhall, R. H., & Langfield-Smith, K. 2007] Robert H. Chenhall and Kim Langfield-Smith. Multiple perspectives of performance measures. *European Management Journal*, *25*(4), 266-282. 2007

[Chiesa, V., et al. 1996] Vittorio Chiesa, Paul Coughlan and Chris A. Voss. Development of a technical innovation audit. *Journal of Product Innovation Management*, *13*(2), 105-136. 1996

[Chrissis, M. B., et al. 2003] Mary Beth Chrissis, Mike Konrad and Sandy Shrum. *CMMi* Addison-Wesley Boston.2003

[Da Silveira, G., et al. 2001] Giovani Da Silveira, Denis Borenstein and Flavio S. Fogliatto. Mass customization: Literature review and research directions. *International Journal of Production Economics*, *72*(1), 1-13. 2001

[Davis, S. M. 1989] S. M. Davis. From "future perfect": Mass customizing. *Strategy & Leadership*, *17*, 1989

[Fitzgerald, L., et al. 1991] Lin Fitzgerald, Stan Brignall, Rhian Silvestro, Christopher Voss and Johnston Robert. *Performance measurement in service businesses* Chartered Institute of Management Accountants London.1991

[Fogliatto, F. S., et al. 2012] F. S. Fogliatto, G. J. C. da Silveira and D. Borenstein. The mass customization decade: An updated review of the literature. *International Journal of Production Economics*, 2012

[Folan, P., & Browne, J. 2005] Paul Folan and Jim Browne. A review of performance measurement: Towards performance management. *Computers in Industry*, *56*(7), 663-680. 2005

[Forza, C., & Salvador, F. 2007] 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.2007

[Franke, N., & Piller, F. T. 2003] Nikolaus Franke and Frank T. Piller. Key research issues in user interaction with user toolkits in a mass customisation system. *International Journal of Technology Management*, *26*(5), 578-599. 2003

[Gregory, M. J. 1993] Mike J. Gregory. Integrated performance measurement: A review of current practice and emerging trends. *International Journal of Production Economics*, *30*, 281-296. 1993

[Heiskala, M., et al. 2010] Mikko Heiskala, Juha Tiihonen, Matti Sievänen and Kaija-Stiina Paloheimo. Modeling concepts for choice navigation of mass customized solutions. *International Journal of Industrial Engineering and Management*, *1*(3), 97-103. 2010

[Huffman, C., & Kahn, B. E. 1998] Cynthia Huffman and Barbara E. Kahn. Variety for sale: Mass customization or mass confusion? *Journal of Retailing*, *74*(4), 491-513. 1998

[Kanji, G. K. 1998] Gopal K. Kanji. Measurement of business excellence. *Total Quality Management*, *9*(7), 633-643. 1998

[Kaplan, R., & Norton, D. 1992] RS Kaplan and DP Norton. The balanced scorecard- measures that drive performance. *Harvard Business Review*, *70*(1), 1992

[Keegan, D. P., et al. 1989] Daniel P. Keegan, Robert G. Eiler and Charles R. Jones. Are your performance measures obsolete? *Management Accounting*, *70*(12), 45-50. 1989

[Kennerley, M., & Neely, A. 2003] Mike Kennerley and Andy Neely. Measuring performance in a changing business environment. *International Journal of Operations & Production Management*, *23*(2), 213-229. 2003

[Kohlegger, M., et al. 2009]Michael Kohlegger, Ronald Maier and Stefan Thalmann. Understanding maturity models results of a structured content analysis. *Proceedings of I-KNOW*, , *9*. pp. 2-4.

[Lynch, R. L., & Cross, K. F. 1992] Richard L. Lynch and Kelvin F. Cross. *Measure up!: The essential guide to measuring business performance* Mandarin.1992

[Lyons, A. C., et al. 2012] A. C. Lyons, A. E. C. Mondragon, F. Piller and R. Poler. Mass customisation: A strategy for customer-centric enterprises. *Customer-Driven Supply Chains*, 2012

[Maier, A. M., et al. 2006] Anja M. Maier, Claudia M. Eckert and P. John Clarkson. Identifying requirements for communication support: A maturity grid-inspired approach. *Expert Systems with Applications*, *31*(4), 663-672. 2006

[Maier, A. M., et al. 2012] Anja M. Maier, James Moultrie and PJohn Clarkson. Assessing organizational capabilities: Reviewing and guiding the development of maturity grids. *Engineering Management, IEEE Transactions On*, *59*(1), 138-159. 2012

[McGrath, M., & Akiyama, C. 1996] ME McGrath and CL Akiyama. PACE: An integrated process for product and cycle time excellence. *Setting the PACE in Product Development, Butterworth and Heinemann, Boston*, , 17-29. 1996

[Medori, D., & Steeple, D. 2000] David Medori and Derek Steeple. A framework for auditing and enhancing performance measurement systems. *International Journal of Operations & Production Management*, *20*(5), 520-533. 2000

[Melnyk, S. A., et al. 2004] Steven A. Melnyk, Douglas M. Stewart and Morgan Swink. Metrics and performance measurement in operations management: Dealing with the metrics maze. *Journal of Operations Management*, *22*(3), 209-218. 2004

[Mortensen, N. H., et al. 2008] N. H. Mortensen, R. Pedersen, M. Kvist and L. Hvam. Modelling and visualising modular product architectures for mass customisation. *International Journal of Mass Customisation*, *2*(3), 216-239. 2008

[Moultrie, J., et al. 2007] James Moultrie, P. John Clarkson and David Probert. Development of a design audit tool for SMEs*. *Journal of Product Innovation Management*, *24*(4), 335-368. 2007

[Neely, A. D., et al. 2002] Andy D. Neely, Chris Adams and Mike Kennerley. *The performance prism: The scorecard for measuring and managing business success* Prentice Hall Financial Times London.2002

[Neely, A., et al. 2005] Andy Neely, Mike Gregory and Ken Platts. Performance measurement system design: A literature review and research agenda. *International Journal of Operations & Production Management*, *25*(12), 1228-1263. 2005

[Nielsen, K., et al. 2012] Kjeld Nielsen, Thomas Ditlev Brunø and Kaj Asbjørn Jørgensen. A FRAMEWORK STUDY ON ASSESSMENT OF MASS CUSTOMIZATION CAPABILITIES.2012

[Partanen, J., & Haapasalo, H. 2004] Jari Partanen and Harri Haapasalo. Fast production for order fulfillment: Implementing mass customization in electronics industry. *International Journal of Production Economics*, *90*(2), 213-222. 2004

[Paulk, M. C., et al. 1993] M. C. Paulk, B. Curtis, M. B. Chrissis and C. V. Weber. Capability maturity model, version 1.1. *Software, IEEE*, *10*(4), 18-27. 1993

[Piller, F., et al. 2012a] F. Piller, E. Lindgens and F. Steiner. Mass customization at adidas: Three strategic capabilities to implement mass customization.2012a

[Piller, F. T., & Tseng, M. M. 2010] Frank T. Piller and Mitchell M. Tseng. *Handbook of research in mass customization and personalization: Strategies and concepts* World Scientific.2010

[Piller, F., et al. 2012b]*Part 7: Overcoming the challenge of implementing mass customization* Innovation Management.

[Pollard, D., et al. 2011] Dennis Pollard, Shirley Chuo and Brian Lee. Strategies for mass customization. *Journal of Business & Economics Research (JBER)*, *6*(7)2011

[Pun, K., & White, A. 2005] KF Pun and AS White. A performance measurement paradigm for integrating strategy formulation: A review of systems and frameworks. *International Journal of Management Reviews*, *7*(1), 49-71. 2005

[Salvador, F., et al. 2009] Fabrizio Salvador, Pablo Martin De Holan and Frank Piller. Cracking the code of mass customization. *MIT Sloan Management Review*, *50*(3), 71-78. 2009

[Szakonyi, R. 1994] Robert Szakonyi. Measuring R&D effectiveness-I. *Research Technology Management*, *37*, 27-27. 1994

[Trentin, A., et al. 2013] Sales configurator capabilities to avoid the product variety paradox: Construct development and validation. *Computers in Industry*, 2013

[Trist, E. 1981] Eric Trist. The evolution of socio-technical systems. *Occasional Paper*, *2*1981

[Walcher, D., & Piller, F. T. 2011] Dominik Walcher and Frank T. Piller. *The customization 500* (1st edition ed.). Aachen: Lulu Press.2011

What makes the Difference? - Basic Characteristics of Configuration

Lothar Hotz
HITeC e.V., University of Hamburg, Germany
hotz@informatik.uni-hamburg.de

Abstract

This paper focuses on configuration as a process that iteratively applies commonly known reasoning techniques and creates an incrementally growing configuration description. This approach emphasizes the synthesis aspect of configuration, which continuously acquires requirements and computes their effects on a configuration in a cyclic way. We provide the definitions of needed ingredients as there are partial configuration, configuration decision, and reasoning for computing entailments of made configuration decisions. These ingredients are the basis for implementations of configuration systems that follow these approach.

1 Introduction

Configuration is the task of composing a valid system description from known component definitions (*configuration model*) and customer requirements. Typical configuration approaches map this task to reasoning techniques such as constraint solving [Sabin and Freuder, 1996; John, 2002], Description Logics [McGuinness, 2003], or Answer Set Programming [Soininen *et al.*, 2001]. Through this mapping, appropriate reasoners solve the configuration task by computing a solution of their respective logical reasoning problems.

These approaches make a fundamental assumption, i.e. that the requirements of the configuration task can be initially given, e.g., through a *a set of requirements* – see also [Sabin and Weigel, 1998] which call this approach *batch configuration*. But for many, not simple, configuration tasks this does not hold [Neumann, 1988; Stumptner *et al.*, 1998; Fleischanderl *et al.*, 1998]. This is due to the fact, that only during the configuration experience, when the configured product grows in front of the user's eye, the user realizes their own desires and needs [Simonson, 2003]. For example, during a sales conversation, if a requirement causes the need of a subsystem, previously not recognized, additional requirements come into account that are related to the subsystem. Thus, the requirements are not completely clear in the beginning but change during the configuration undertaking. Simple examples are web-based configurators for consumer products such as cars or electronic items that lead the user through a sequence of web-pages for successively acquiring requirements. Other examples are industrial configurators which firstly acquire features of a system and than configure system specific components, like it is described in [Haag, 1998; Ranze *et al.*, 2002; Hotz *et al.*, 2006] – see also [Sabin and Weigel, 1998] which call this approach *incremental configuration*.

These considerations lead to a configuration approach that combines reasoning techniques with the characteristic of a *configuration process*. Process-related subtasks are identification of next steps in the configuration process or managing partial configurations. Especially the retraction of decisions previously made by a user is a characteristic of configuration processes [Günter and Cunis, 1992; Hotz and Wolter, 2013].

Configuration approaches that take only a certain reasoning technology into account, such as configuration based on Description Logics [McGuinness, 2003] or pure constraint processing [Tsang, 1993], have to build an external architecture (or even user interface) around the reasoning kernel, which handles configuration process tasks. Such approaches consider a configuration process as a sequence of changing but fully defined configuration tasks. However, they leave the process-related subtasks to the external architecture and do not integrate them in a configuration system. These approaches lead to domain dependent non-declarative implemented configuration processes.

If a configuration system integrates reasoning facilities and process or procedural aspects, such as e.g. [Günter and Cunis, 1992; Stumptner *et al.*, 1998; Fleischanderl *et al.*, 1998; Günter and Hotz, 1999], the inherent dynamic aspects of configuration tasks can be solved in a general, domain independent way, based on the configuration model. This view is also supported by the early definitions or thoughts about configuration as a syntheses task, in opposite to an analysis task like diagnosis [Brown and Chandrasekaran, 1989; Cunis *et al.*, 1989; Günter and Cunis, 1992; Brown, 1996; Günter and Kühn, 1999].

Other approaches that focus on such aspect of configuration are generative constraints. In [Stumptner *et al.*, 1998] the incrementing configuration is modeled through specific variables that are activated if certain parts of a configured system come into play. This approach is similar to the one defined in the following, however, we do not map the generative notion to a certain reasoning technology (such as constraints in

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

[Stumptner *et al.*, 1998]), but we provide a framework around a reasoning technology that applies it on subsequently defined new configuration tasks. This is done by iteratively including requirement acquisition in the configuration process. Furthermore, we will not concentrate on a certain reasoning technique, but provide a general framework.

Thus, in this position paper, we elaborate on basic characteristics of the configuration process which lead to a complex mapping of the configuration task to reasoning techniques. Newly introduced operators allow a definition of the dynamics of the configuration process. These include definitions for components and restrictions as well as requirements as usual. Additionally, they provide notions for partial configurations and requirement variables. Furthermore, operators for computing requirement variables, acquiring requirements, and for the actual reasoning will lead to a comprehensive definition of the configuration problem.

Thus, we focus on the iterative characteristic of configuration, i.e. incrementing configuration with intermediate partial configurations (see Section 2). By taking this view, the paper furthermore tries to clarify the relationship of configuration to other reasoning techniques (see Section 3).

We follow ideas published in [Cunis *et al.*, 1989; Günter, 1995]. However, by introducing process related definitions and operators, a summary of the needed ingredients is established that shall give the basis for process-based configuration technologies.

2 General Definitions

In the following, through definitions that build on each other, we develop a definition of a complex configuration problem that take the iterative character of the configuration process into account. We develop a general definition of the complex configuration task that is independent of a certain knowledge representation, such as logic or constraint-based approaches.

First, we provide the definition of a configuration model. This model defines all possible configurations in a generic way. The model represents entities to be configured (such as hardware components, software modules, or services) and relations between them.

We here discuss a combination of entities (e.g., represented with component types or classes) and relations (e.g., represented with constraints). However, the operators defined in the following are independent of the representation. For example, if a pure constraint-based representation is initially used, appropriate operators should have to be developed for supporting the here considered complex configuration tasks.

The definitions are illustrated with an example of composing a menu with antipasti, main course, and dessert. While antipasti will always be selected if a hearty menu is desired, the selection of a dessert cannot be computed from constraints. A hearty menu is part of initial requirements, while the dessert is only selected after the main course, demonstrating a dynamic requirements acquisition.

Definition 1 (Configuration Model)**.** A configuration model $\mathcal{CM}$ is a generic description of entities of an application domain. $\mathcal{CM}$ is a tuple $\langle \Gamma, \Psi, \Phi \rangle$, where

- Γ is a set of attributed entity models $\mathcal{EM} \in \Gamma$ (e.g. classes or concepts) each representing a set of concrete entities to be configured. An entity model consists of named properties. Each property $\mathcal{P}$ is a binary relation that maps from $\mathcal{EM}$ to a *property domain* $\mathcal{PD}_\mathcal{P}$.
- Ψ is a set of property domains $\mathcal{PD}_\mathcal{P} \in \Psi$ of properties $\mathcal{P}$. A $\mathcal{PD}_\mathcal{P}$ might be a *structural property domain* (and $\mathcal{P}$ is called *structural property*) with a set of *structural property value*s, i.e. an entity model optionally combined with a cardinality. Or it might be a primitive datatype, such as a number, symbol, or string or a probably infinite set (e.g. for number ranges) of those, than $\mathcal{P}$ is called *data-type property*.
- Φ is a set of n-ary relations $\mathcal{ER} \in \Phi$ between properties of entity models.

Example 1. *An entity model with one data-type property, one structural property, and a n-ary relation representing the fact that a menu should consist in any case of a main course and might optionally has an antipasti and a dessert. Depending on the kind of taste, an antipasti is selected if hearty taste is desired:*

```
(Entity-Model name: Menu
  super-type: Aggregate
  data-properties: ((kindOfTaste {hearty light}))
  hasCourses:
   ((AntiPasti :min 0 :max 1)
    (MainCourse :min 1 :max 1)
    (Dessert :min 0 :max 1)))
(Entity-Model name: AntiPasti
  super-type: Part)
(Entity-Model name: MainCourse
  super-type: Part)
(Entity-Model name: Dessert
  super-type: Part)

(Constraint name: HeartyEqualsToAntiPasti
  Menu.kindOfTaste == hearty <=>
  Menu.hasCourses HAS (AntiPasti :min 1 :max 1))
```

For representing concrete components of an application domain, *entity instances* are used. First, we define a simple entity instance, later on we enhance this definition.

Definition 2 (Simple Entity Instance)**.** A $\mathcal{SEI}_{\mathcal{EM}}$ represents one concrete entity of the application domain. $\mathcal{SEI}_{\mathcal{EM}}$ belongs to an entity model $\mathcal{EM}$ and has all or some properties of $\mathcal{EM}$. The property values are constant values that belong to the respective property values of $\mathcal{EM}$.

For representing customer requirements about the desired configuration, configuration requirements are defined as follows.

Definition 3 (Simple Configuration Requirememts)**.** Simple configuration requirements $\mathcal{SR}$ are a set of simple entity instances with some properties filled with constant values.

Example 2. *Two simple entity instances representing the requirement "Hearty menu with a main course":*

```
(Entity-Instance name: menu-1
  entity-model: Menu
  kindOfTaste: {hearty}
  hasCourses:  {mainCourse-1})
(Entity-Instance name: mainCourse-1
  entity-model: MainCourse)
```

Typically, a configuration task is defined as follows:

Definition 4 (Simple Configuration Task)**.** A configuration task is defined through an entity model $\mathcal{EM}$ and configuration requirements $\mathcal{SR}$.

Definition 5 (Simple Configuration)**.** A configuration is defined through a set of completely filled configuration instances $\mathcal{CI}$.

A knowledge representation technique allows it to represent an configuration model and configuration requirements. Furthermore, it provides reasoning techniques that allow to solve the simple configuration problem.

Definition 6 (Simple Configuration Problem)**.** $\mathcal{CM} = \langle \Gamma, \Psi, \Phi \rangle$ be a configuration model. A *simple configuration problem* in $\mathcal{CM}$ is a tuple $\langle \mathcal{CM}, \mathcal{SR} \rangle$, where $\mathcal{SR}$ is a set of initial simple entity instances. A *solution* (a *configuration*) of the problem $\langle \mathcal{CM}, \mathcal{SR} \rangle$ is a set of simple entity instances that is consistent with $\mathcal{CM}$ and fulfills the configuration requirements $\mathcal{SR}$.

How consistency is concretely defined depends on the knowledge representation. However, in general for structural properties, consistency means that entity instances belong to $\mathcal{S}$ according to the cardinality descriptions of the structural property. Thus, structural relations emphasize a configuration being a collection of related entity instances, while n-ary relations related properties of those instances.

Example 3. *A resulting configuration, the constraint infers the need of an entity instance representing antipasti:*

```
(Entity-Instance name: menu-1
  entity-model: Menu
  kindOfTaste: {hearty}
  hasCourses:  {mainCourse-1 antiPasti-1})
(Entity-Instance name: mainCourse-1
  entity-model: MainCourse)
(Entity-Instance name: antiPasti-1
  entity-model: AntiPasti)
```

These definition provide the basis for configuration tasks: a configuration model, the customer requirements, and a knowledge representation that allows for creating a configuration that fulfills the customer requirements. This definition makes a basic assumption, i.e. that $\mathcal{SR}$, the set of particular customer requirements, are given. In simple, one step configuration problems, such as parameterization of technical systems, this is a reasonable assumption. In more complex applications, the requirements evolve during the configuration process. This observation leads to further definitions.

First, we enhance the definition of a simple entity instance by allowing not only constant values for properties but also subsets.

Definition 7 (Partial Property Domain)**.** Let $\mathcal{P}$ be a property of an entity instance $\mathcal{EI}$ of entity model $\mathcal{EM}$. And let $\mathcal{PD}_\mathcal{P}$ be the property value of $\mathcal{P}$ as defined in $\mathcal{CM}$. A partial property domain $\mathcal{PPD}_\mathcal{P}$ of $\mathcal{P}$ is a subset of $\mathcal{PD}_\mathcal{P}$, i.e. $\mathcal{PPD}_\mathcal{P} \subseteq \mathcal{PD}_\mathcal{P}$.

Please note that a specific partial property domain is a property domain with one value representing a constant value, e.g. a number of a data-type property or a single instance of a structural property.

Example 4. *Example for a partial property domain of a structural property with one entity instance and two open cardinality definitions:*

```
hasCourses: {mainCourse-1 (AntiPasti :min 0 :max 1)
                          (Dessert :min 0 :max 1)}
```

Definition 8 (Terminal Property Domain)**.** A terminal property domain $\mathcal{TPD}_\mathcal{P}$ of property $\mathcal{P}$ is a partial property domain that is marked with *terminal*. A constant value is automatically marked as terminal. Structural property values or sets may be marked through the heuristic operator (see below).

Example 5. *Example for a partial property domains of a data-type property indicated as terminal:*

```
kindOfTaste: {hearty [terminal]}
```

Definition 9 (Entity Instance)**.** An $\mathcal{EI}_\mathcal{EM}$ represents one concrete entity of the application domain. $\mathcal{EI}_\mathcal{EM}$ belongs to an entity model $\mathcal{EM}$ and has the same properties as $\mathcal{EM}$. The property values might be the same as defined for $\mathcal{EM}$ or subsets of those, they might be partial or terminal property domains. These *partially filled entity instances* represent uncertain knowledge about the concrete entity. A *completely filled entity instance* has a terminal property domain for *each* property.

Example 6. *One partially filled entity instance:*

```
(Entity-Instance name: menu-1
  entity-model: Menu
  kindOfTaste: hearty
  hasCourses:  {mainCourse-1 (AntiPasti :min 0 :max 1)
                             (Dessert :min 0 :max 1)})
```

Example 7. *One completely filled entity instance:*

```
(Entity-Instance name: menu-1
  entity-model: Menu
  kindOfTaste: {hearty [terminal]}
  hasCourses:  {mainCourse-1 antiPasti-1 [terminal]})
```

Definition 10 (Partial Configuration)**.** A partial configuration $\mathcal{PC}$ is a set of partially or completely filled entity instances.

Definition 11 (Configuration Requirememts)**.** Configuration requirements $\mathcal{R}$ are a set of instances with some property values set to terminal property domains.

Thus, configuration requirements are a specific kind of partial configuration namely one with instances whose properties do not have a terminal property domain for each property. We also call this partial configuration *initial partial configuration*.

Definition 12 (Final Configuration)**.** A final configuration $\mathcal{FC}$ is a set of completely filled entity instances. Furthermore, for each structural property of an instance in $\mathcal{FC}$, a related instances exists in $\mathcal{FC}$ according to the structural property value (i.e. the defined entity model and the cardinality).

Now, probably the main step in our definitions follows, i.e. the introduction of the definition of a variable. Typically, properties of components are considered as variables and the configuration task is to provide a value for these variables, i.e. for the properties of the components. In our definition, a variable stands for a decision that has to be made for gaining a final configuration. Thereby, each variable stands for one property of an entity instance that has to be determined. However, during the configuration process there might be several variables for one property, e.g. if a property value is reduced by the user in several steps.

Definition 13 (Variable)**.** A variable $\mathcal{V}$ represents one decision of setting the value of one certain property. One or more decisions have to be made for each property of every entity instance. The variable represents possible outcomes of the decision through its variable domain $\mathcal{V}_d$. A variable domain is a property value.

Example 8. *Variable representing the decision that antipasti and dessert shall be selected as courses of a menu:*

```
(Variable
 entity-instance: menu-1
 property: hasCourses
 property-value: {mainCourse-1
                  (AntiPasti :min 0 :max 1)
                  (Dessert :min 0 :max 1)})
```

Definition 14 (Reduced Variable)**.** A reduced variable $\mathcal{R}_V$ of a variable $\mathcal{V}$ with a domain $\mathcal{V}_d$ is a variable with a domain $\mathcal{R}_{V_d}$ with $\mathcal{R}_{V_d} \subset \mathcal{V}_d$. The reduced domain might also be a terminal property domain. For a structural property with a structural property value, the subset is a set of instances that are conform with the cardinality descriptions of the structural property value.

The reduced variable represents the result of a made decision.

Definition 15 (Heuristic Operator)**.** A heuristic operator $\mathcal{HO}$ is an operator that takes a variable $\mathcal{V}$ and computes a reduced variable $\mathcal{R}_\mathcal{V}$ (probably with a terminal property domain) by some heuristic method, thus: $\mathcal{HO}$: $\mathcal{V} \rightarrow \mathcal{R}_\mathcal{V}$.

The heuristic operator represents the method for gaining a reduced variable, e.g. the selection of a default value, the computation of a function computing a reduced domain for the variable, or *the acquisition of a value for that variable from the user*. Thus, the heuristic operator acquires subsequent requirements that come up during the configuration process. Furthermore, by reducing a structural property the heuristic operator incrementally expands the configuration, because new entity instances are created when reducing the structural property (see above). A new entity instance has the same properties and property values as its entity model.

Example 9. *Reduced variable created by a heuristic operator that asks the user, if a dessert is needed, answer was "yes":*

```
(Variable
 entity-instance: menu-1
 property: hasCourses
 property-value: {mainCourse-1
                  (AntiPasti :min 0 :max 1)
                  dessert-1)})
```

Example 9 represents the answer to the typically raised question after a meal "Would you like a dessert?", which is a simplistic example for a dynamic requirement acquisition.

Definition 16 (Entailment Operator)**.** An entailment operator $\mathcal{EO}$ is an operator that takes the configuration model $\mathcal{CM}$ (especially the defined n-ary relations), a partial configuration $\mathcal{PC}_i$, and one reduced variable $\mathcal{R}_\mathcal{V}$ and computes a new partial configuration $\mathcal{PC}_{i+1}$ which contains the value of the reduced variable and all entailments of this reduction, computed by some reasoning method, thus:
$\mathcal{EO}$: $\mathcal{CM}$, $\mathcal{PC}_i$, $\mathcal{R}_\mathcal{V} \rightarrow \mathcal{PC}_{i+1}$.
In the initial case, $\mathcal{R}_\mathcal{V}$ might be empty, i.e. $\mathcal{EO}$: $\mathcal{CM}$, $\mathcal{PC}_0$, $\rightarrow \mathcal{PC}_1$ computing the entailments of the values in $\mathcal{PC}_0$, i.e. the initial partial configuration.

The entailment operator represents the integration of one made decision into a partial configuration and the computation of the influences of this decision to the partial configuration. The influences are computed on the basis of the configuration model, especially the n-ary relations, which relate properties of entity models. An influence or entailment is a reduction of domains of some variables in $\mathcal{PC}_i$. Typical examples for an entailment operator are the solution of a constraint problem or applying Description Logic services.

Example 10. *Reduced variable created by an entailment operator that uses the constraint for deciding that an antipasti is needed if a hearty menu was selected:*

```
(Variable
 entity-instance: menu-1
 property: hasCourses
 property-value: {mainCourse-1
                  antiPasti-1
                  (Dessert :min 0 :max 1)})
```

Definition 17 (Open Issue Operator)**.** An open issue operator $\mathcal{OIO}$ is an operator that takes the configuration model $\mathcal{CM}$ and a partial configuration $\mathcal{PC}$ and computes new variables $\mathcal{V}_i$ that have no terminal property domains and are collected in the set $\mathcal{A}$, thus: $\mathcal{OIO}$: $\mathcal{CM}$, $\mathcal{PC} \rightarrow \mathcal{A}$, with $\mathcal{V}_i \in \mathcal{A}$.

From Example 6 the $\mathcal{OIO}$ computes the variable shown in Example 8 because of the partial property domain for property `hasCourses`.

Definition 18 (Select Operator)**.** A select operator $\mathcal{SO}$ is an operator that select one variable $\mathcal{V}$ out of the set $\mathcal{A}$ by some selection method, thus: $\mathcal{SO}$: $\mathcal{A} \rightarrow \mathcal{V}$.

Now, we define the actual configuration process and identify its main part, i.e. the *configuration cycle*. A configuration process starts with an initial partial configuration given through the requirements of a customer. By successively applying the above operators the final configuration will be created. This process is defined as follows:

Starting from the initial partial configuration $\mathcal{IP}$, $\mathcal{EO}$ computes the first entailments and, thus, $\mathcal{PC}_1$. Hereafter, the open issue operator $\mathcal{OIO}$ can be applied to $\mathcal{PC}_1$ for computing next variables with non-terminal property domains and builds $\mathcal{A}_1$. The operator $\mathcal{SO}$ selects a next variable to be decided $\mathcal{V}_1$. From here the heuristic operator $\mathcal{HO}$ reduces the variable's domain to $\mathcal{RV}_1$. The entailment operator uses the previous partial configuration ($\mathcal{EO}(\mathcal{PC}_1)$) for computing the next partial configuration $\mathcal{PC}_2$. This way the operators are successively applied until the final configuration $\mathcal{PC}_n$ is created and no more open issues can be identified (i.e. $\mathcal{OIO}$ computes an empty set). In total, we have:

$$\mathcal{IP}, \xrightarrow{\mathcal{EO}} \mathcal{PC}_1 \xrightarrow{\mathcal{OIO}} \mathcal{A}_1 \xrightarrow{\mathcal{SO}} \mathcal{V}_1 \xrightarrow{\mathcal{HO}} \mathcal{RV}_1$$
$$\xrightarrow{\mathcal{EO}(\mathcal{PC}_1)} \mathcal{PC}_2 \xrightarrow{\mathcal{OIO}} \mathcal{A}_2 \xrightarrow{\mathcal{SO}} \mathcal{V}_2 \xrightarrow{\mathcal{HO}} \mathcal{RV}_2$$
$$\xrightarrow{\mathcal{EO}(\mathcal{PC}_2)} \mathcal{PC}_3 \xrightarrow{\mathcal{OIO}} \mathcal{A}_3 \xrightarrow{\mathcal{SO}} \mathcal{V}_3 \xrightarrow{\mathcal{HO}} \mathcal{RV}_3$$
$$\ldots$$
$$\xrightarrow{\mathcal{EO}(\mathcal{PC}_{n-2})} \mathcal{PC}_{n-1} \xrightarrow{\mathcal{OIO}} \mathcal{A}_{n-1} \xrightarrow{\mathcal{SO}} \mathcal{V}_{n-1} \xrightarrow{\mathcal{HO}} \mathcal{RV}_{n-1}$$
$$\xrightarrow{\mathcal{EO}(\mathcal{PC}_{n-1})} \mathcal{PC}_n \xrightarrow{\mathcal{OIO}} \emptyset$$

Thus, the general configuration cycle is defined as follows:

Definition 19 (Configuration Cycle)**.** A configuration cycle $\mathcal{COC}$ is a sequence of operators $\mathcal{OIO}, \mathcal{SO}, \mathcal{HO}, \mathcal{EO}$ as follows:

$$\mathcal{PC}_i \xrightarrow{\mathcal{OIO}} \mathcal{A}_i \xrightarrow{\mathcal{SO}} \mathcal{V}_i \xrightarrow{\mathcal{HO}} \mathcal{RV}_i \xrightarrow{\mathcal{EO}(\mathcal{PC}_i)} \mathcal{PC}_{i+1}$$

Definition 20 (Complex Configuration Problem)**.** $\mathcal{CM} = \langle \Gamma, \Psi, \Phi \rangle$ be a configuration model. A *complex configuration problem* in $\mathcal{CM}$ is a tuple $\langle \mathcal{CM}, \mathcal{R}, \mathcal{HO}, \mathcal{EO}, \mathcal{SO}, \mathcal{OIO} \rangle$, where $\mathcal{R}$ is a set of initial entity instances and $\mathcal{HO}, \mathcal{EO}, \mathcal{SO}, \mathcal{OIO}$ the previously introduced operators. A *solution* of the problem $\langle \mathcal{CM}, \mathcal{R}, \mathcal{HO}, \mathcal{EO}, \mathcal{SO}, \mathcal{OIO} \rangle$ is a final configuration that was computed by applying the configuration cycle, that is consistent with $\mathcal{CM}$, and that fulfills the configuration requirements $\mathcal{R}$.

Like in the simple configuration problem definition, how consistency is concretely defined depends on the knowledge representation. Furthermore, the knowledge representation defines the entailment operator.

3 Discussion

The commonly used view on configuration is to specify a configuration model and customer requirements as a reasoning task of a reasoning system, such as a constraint system, and than solve the reasoning task and present the resulting configuration. This paper provides a view on configuration that basically iterates these two steps of defining requirements and reason about them, i.e.:

1. Start from an initial configuration including the requirements,
2. compute entailment of the requirements on the configuration (operator $\mathcal{EO}$),
3. create an agenda with not yet made decisions (operator $\mathcal{OIO}$),
4. select one decision (operator $\mathcal{SO}$),
5. make the decision (operator $\mathcal{HO}$), and
6. compute its entailments (operator $\mathcal{EO}$),
7. goto Step 3.

The computation of the entailments (Step 2 and Step 6) correspond to the solution of the reasoning task, e.g. by constraint processing. The construction of a reasoning task is included in the configuration process in the steps 3, 4, and 5.

Thus, the overall schema for configuration as seen in our approach provides an iterative application of commonly applied reasoning techniques for configuration. For solving a complex configuration problem, a configuration system or a technological approach should realize the operators defined above.

Of course, there exist variations of the here presented basic framework. For example, the proposed approach to represent requirements with instances might be enhanced to complex requirements that need further reasoning to compute them [Thäringen, 1995; Kopisch and Günter, 1992]. Or the selection of a decision may be enhanced to selecting multiple decisions or to let the user select next decisions from the agenda. Another extension is to include techniques for conflict resolution [Günter and Hotz, 1995; Felfernig and Schubert, 2010; Hotz and Wolter, 2013]. However, this paper provides the basic ingredients for solving a configuration task incrementally. Configuration tools which follow our approach are KONWERK [Günter and Hotz, 1999], engcon [Hollmann *et al.*, 2000], or Plakon [Cunis *et al.*, 1989].

4 Summary

This paper defines the necessary ingredients for a configuration process that iteratively generates a configuration. Besides the typically used reasoning techniques, the process additionally supplies steps for creating requirements on the fly and include them and their entailments in a growing configuration. Enhancements in future work will be the inclusion of operators for resolving conflicts that might occur during the configuration process.

References

[Brown and Chandrasekaran, 1989] D.C. Brown and B. Chandrasekaran. *Design Problem Solving - Knowledge Structures and Conrtol Strategies*. Research Notes in Artificial Intelligence Series. Pitman Publishing, London, 1989.

[Brown, 1996] D.C. Brown. Some Thoughts on Configuration Processes. AAAI 1996 Fall Symposium Workshop: Configuration FS-96-03, MIT, Cambridge, Massachusetss, USA, 1996.

[Cunis *et al.*, 1989] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON - An Approach to Domain-Independent Construction. In *Proc. of Second Int. Conf. on Industrial and Engineering Applications of AI and Expert Systems IEA/AIE-89*, pages 866–874, June 6-9 1989.

[Felfernig and Schubert, 2010] A. Felfernig and M. Schubert. Diagnosing Inconsistent Requirements. In L. Hotz and A. Haselböck, editors, *Proc. of the Configuration Workshop on 19th European Conference on Artificial Intelligence (ECAI-2010)*, Lisbon, Portugal, August 2010.

[Fleischanderl *et al.*, 1998] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, July/August 1998.

[Günter and Cunis, 1992] A. Günter and R. Cunis. Flexible Control in Expert Systems for Construction Tasks. *Journal Applied Intelligence*, 2(4):369–385, 1992.

[Günter and Hotz, 1995] A. Günter and L. Hotz. Auflösung von Konfigurationskonflikten mit Wissensbasiertem Backtracking und Reparaturanweisungen (Conflict Resolution with Knowledge-based Backtracking and Repair-Statement). In A. Günter, editor, *"Wissensbasiertes Konfigurieren"*, St. Augustin, 1995. Infix.

[Günter and Hotz, 1999] A. Günter and L. Hotz. KONWERK - A Domain Independent Configuration Tool. *Configuration Papers from the AAAI Workshop*, pages 10–19, July 1999.

[Günter and Kühn, 1999] A. Günter and C. Kühn. Knowledge-Based Configuration - Survey and Future Directions. In F. Puppe, editor, *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg, March 3-5 1999.

[Günter, 1995] A. Günter. *Wissensbasiertes Konfigurieren (Knowledge-based Configuration)*. Infix, St. Augustin, 1995.

[Haag, 1998] A. Haag. Sales Configuration in Business Processes. *IEEE Intelligent Systems*, pages 78–85, July August 1998.

[Hollmann *et al.*, 2000] O. Hollmann, T. Wagner, and A. Günter. EngCon: A Flexible Domain-Independent Configuration Engine. In *Proc. ECAI-Workshop Configuration*, page 94 pp, Berlin, Germany, August 21-22 2000.

[Hotz and Wolter, 2013] Lothar Hotz and Katharina Wolter. Beyond Physical Product Configuration - Configuration in Unusual Domains. *AI Commun.*, 26(1):39–66, 2013.

[Hotz *et al.*, 2006] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor. *Configuration in Industrial Product Families - The ConIPF Methodology*. IOS Press, Berlin, 2006.

[John, 2002] U. John. *Konfiguration und Rekonfiguration mittels Constraint-basierter Modellierung (Configuration and Reconfiguration by Means of Constraint-Based Modeling)*. Infix, St. Augustin, 2002.

[Kopisch and Günter, 1992] M. Kopisch and A. Günter. Configuration of a Passenger Aircraft Cabin - based on Conceptual Hierarchy, Constraints and Flexible Control. In F. Belli and F.J. Radermacher, editors, *Proceedings of IEA/AIE*, Paderborn, 1992. Springer-Verlag.

[McGuinness, 2003] D. L. McGuinness. Configuration. In Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *Description Logic Handbook*, pages 397–413. Cambridge University Press, 2003.

[Neumann, 1988] B. Neumann. Configuration Expert Systems: A Case Study and Tutorial. In Bunke, editor, *Proc. 1988 SGAICO Conference on Artificial Intelligence in Manufacturing, Assembly, and Robotics*. Oldenbourg, Munich, 1988.

[Ranze *et al.*, 2002] K.C. Ranze, T. Scholz, T. Wagner, A. Günter, O. Herzog, O. Hollmann, C. Schlieder, and V. Arlt. A Structure-Based Configuration Tool: Drive Solution Designer DSD. *14. Conf. Innovative Applications of AI*, 2002.

[Sabin and Freuder, 1996] D. Sabin and E.C. Freuder. Configuration as Composite Constraint Satisfaction. In *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161. AAAI Press, 1996.

[Sabin and Weigel, 1998] Daniel Sabin and Reiner Weigel. Product Configuration Frameworks - A Survey. *IEEE Intelligent Systems*, pages 42–49, 1998.

[Simonson, 2003] I. Simonson. Determinants of Customer's Responses to Customized Offers: Conceptual Framework and Research Propositions. *Stanford GSB Working Paper No. 1794*, 2003.

[Soininen *et al.*, 2001] Timo Soininen, Ilkka Niemelä, Juha Tiihonen, and Reijo Sulonen. Representing Configuration Knowledge with Weight Constraint Rules. In Alessandro Provetti and Tran Cao Son, editors, *1st International Workshop on Answer Set Programming: Towards Efficient and Scalable Knowledge*, pages 195–201, 2001.

[Stumptner *et al.*, 1998] M. Stumptner, G. Friedrich, and A. Haselböck. Generative Constraint-based Configuration of Large Technical Systems. *AI EDAM*, 12(04):307–320, 1998.

[Thäringen, 1995] M. Thäringen. Wissensbasierte Erfassung von Anforderungen (Knowledge-based Acquisition of Requirements). In A. Günter, editor, *Wissensbasiertes Konfigurieren*. Infix, 1995.

[Tsang, 1993] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, San Diego, New York, 1993.

(Re-)configuration of Communication Networks in the Context of M2M Applications

Iulia Nica and Franz Wotawa
Institute for Software Technology
Graz University of Technology, Graz, Austria
{inica,wotawa}@ist.tugraz.at

Abstract

Machine-to-machine (M2M) communication is of increasing importance in industry due to novel applications, i.e., smart metering or tracking devices in the logistics domain. These applications provoke new requirements for mobile phone networks, which have to be adapted in order to meet these future requirements. Hence, reconfiguration of such networks depending on M2M application scenarios is highly required. In this paper, we discuss modeling for reconfiguration of mobile phone networks in case of M2M applications and present the foundations behind our tool including the used modeling language and the reconfiguration algorithm.

1 Introduction

Because of the availability of communication networks almost everywhere new applications arise. Besides mobile phones for accessing the internet or simple making phone calls, machine-to machine (M2M) communication becomes a still growing application domain for mobile phone networks. Connecting home appliances like alarm systems or heating installations remotely is one application area. Others are tracking of goods in the logistics domain and smart metering. The latter deals with metering of electrical power or water consumption of homes on a fine granular basis of minutes or hours instead of months or even years. In many countries the administration enforces the use of smart metering in order to rise the customer's awareness of their current consumptions in order to reduce the need for electricity, water or other resources.

Besides this educational effect, there are other advantages of smart metering systems. The overall costs for metering might be reduced because the more labor intensive manual metering is not longer necessary. The supplier of resources gains more information regarding current consumptions, which likely improves prediction of consumptions and further allows for improving the stability of the overall supply network. This is especially important for power networks where electricity has to be generated when needed. Unfortunately, power plants cannot be turned on or off without a substantial delay. Another advantage is that the supplier gains direct remote access to the interface between the network and customer. This allows for instance turning off consumer loads whenever needed in order to prevent for example from a blackout.

M2M communication has been a growing market that causes more and more communication over mobile phone networks. Hence, mobile phone companies providing the infrastructure have to adapt their networks due to future needs. Moreover, a M2M application provider has to be ensured that the current mobile phone network is capable of providing enough resource in order to carry out communication requirements. Within the *Simulation and Configuration of Mobile Networks with M2M Applications (SIMOA)* project the objective has been to develop a simulation and reconfiguration environment for smart metering applications in order (1) to ensure that current mobile phone networks are capable of providing enough resources (through simulation), and (2) to give advice for changing either the smart metering solution or the communication network in cases of lack of resources (through reconfiguration). Hence, we first simulate a network configuration, in order to check whether this can support the set of user requirements, and, if the simulation fails (the requirements can not be fulfilled by the mobile network), a working reconfiguration of the given system has to be delivered.

In this paper, we focus on the SIMOA approach to reconfiguration comprising the modeling language SIMOL, which is an object-oriented language, and the reconfiguration engine that makes use of constraint solving and ideas from diagnosis in order to compute system changes. The key concept of SIMOL is the definition of basic and hierarchical components, which are used to represent the desired system. The behavior of a component has to be provided as set of equations. If a component is a subclass of another component, the equations of the superclass are taken together with the equations of the component in order to specify the component's behavior. In SIMOL, it is also possible to assign equations to specific behavior modes. Such a mode might represent a potential configuration like stating a component to be active or inactive in a certain configuration. Alternatively, a mode might represent the range of values assigned to a certain parameter.

Another feature of SIMOL is its ability to model the systems behavior over time. In this case, SIMOL allows for spec-

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

ifying state transfer functions. Such a function itself is a set of equations, that connect values of variables between one state and its successor state. SIMOL does not allow to deal with continuous time. Only systems which can be modeled using discrete time can be represented in SIMOL. The decision to restrict modeling to discrete time via states is due to the requirements of the smart metering application domain, where not the continuous evolution of parameters is important, but the their discrete values. Besides the model of the system, also the system's requirements can be modeled using SIMOL. Hence, every information needed for stating a reconfiguration problem can be formalized using SIMOL.

The reconfiguration algorithm is based on model-based diagnosis, where the idea is to select one mode for each component such that all system's requirements are fulfilled. This problem can be easily stated as constraint satisfaction problem. Hence, we make use of an available constraint solver for computing simulations and reconfigurations. In the following, we discuss the whole SIMOA system architecture, the SIMOL language, and the reconfiguration algorithm in more detail. A discussion on related research and a summary of the content conclude this paper.

2 The SIMOA architecture

In Figure 1, we depict the SIMOA system architecture. The architecture comprises at the highest level two parts: a graphical user interface (*SIMOA M2M GUI*) and the configuration core (*ConfigCore*). The latter is general and can be used in various applications, whereas the other is application specific and has to be tailored accordingly to the requirements. The configuration core itself comprises a compiler that translates models written in SIMOL into MINION constraints [Jefferson *et al.*, 2012; Gent *et al.*, 2006]. MINION is a constraint solver coming with its own constraint language, which is not easily accessible for non-experts in constraint solving. The reasons for choosing MINION were the easy integration into the program written in Java and the very good reasoning performance (being able to solve 8,000 constraints in less than 2 seconds). The MINION program is used in the reconfiguration engine *ReConf* together with the MINION constraint solver to compute valid reconfigurations, which are given back as *Results*.

The interface between the graphical user interface and the configuration core is represented by the SIMOL program and the results obtained from *ReConf*. The SIMOL program comprises the information necessary to specify the system to be reconfigured and the given pre-specified requirements. The reconfiguration result is basically nothing else than a set of possible component modes, that are necessary to fulfill the requirements, together with the computed values for the attributes. The presentation of these results to the user has to be implemented in the user interface and is application specific. In Figure 2, the current user interface of the SIMOA M2M application is given. This graphical user interface enables the user to specify a smart metering application, by placing the meters as well as the cells, which provide access to the mobile phone network, among other components at the appropriate positions. Moreover, the user might specify additional

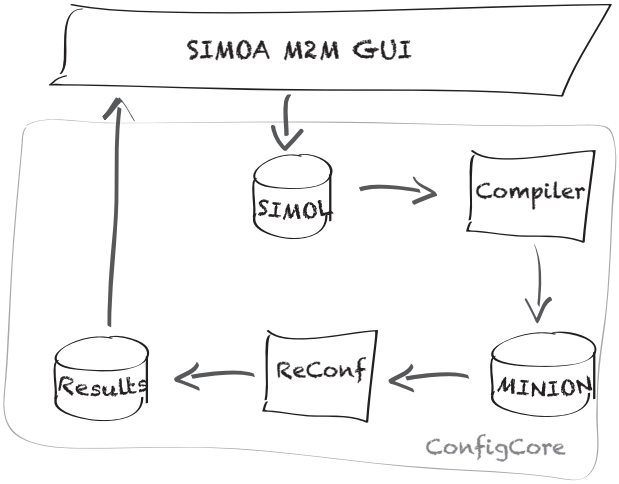


Figure 1: The SIMOA architecture

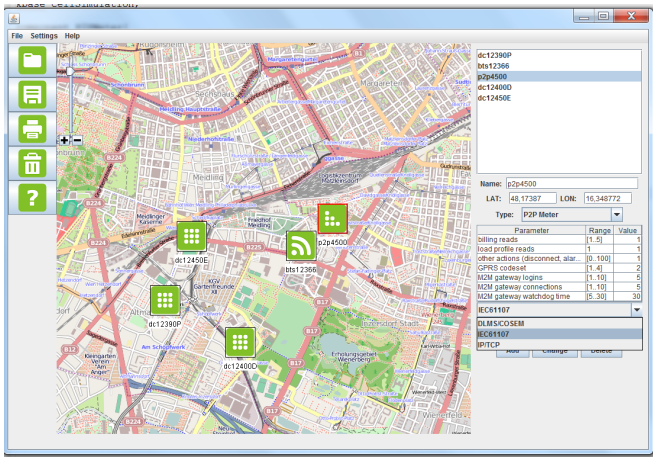


Figure 2: The M2M user interface for the smart metering application

parameters for components. In case of a reconfiguration, the GUI generates a SIMOL program that makes use of the components, their behavior and additional parameters. It is also worth noting that also the positions of the components in the map are used. For example, when specifying a base load (that represents all the non-smart-metering traffic) for the mobile phone network, the concrete assignment to cells is done considering the distance between the base load and the cell. If a base load is not within reach, there is no effect. If a base load might influence two or more cells, the load is assigned to each cell accordingly to their distance. For example, closer cells will have a larger percentage of the communication base load than cells that are more far away.

The *ConfigCore* of the SIMOA architecture is general and can be used in various reconfiguration applications. In this respect, we have conducted a series of experiments using, for instance, combinational circuits from the well-known ISCAS85 benchmark suite. Due to limitations of the *ReConf*

part, we do not handle generative constraints. Hence, building systems from scratch using the given constraints is not possible in SIMOA. However, to some extent, configuration of systems is possible by providing a model of a larger system, where parts can be activated or inactivated. In our application domain there is no impact due to restrictions, because the network, as well as the M2M application structure, are already known and only small deviations are possible. Therefore, providing information regarding structural system changes and changes in parameter is sufficient. In the next section, we discuss SIMOL in more detail. Moreover, we introduce a basic algorithm for reconfiguration using SIMOL programs. For more information regarding the application domain we refer the interested reader to [Nica *et al.*, 2012].

```
1.  kbase GPRSCell;
2.  component P2PMeter {
3.    attribute int mdist,codeset,mRate;
4.    constraints {
5.      mdist = {1..3};
6.      codeset = {1..4};
7.    }
8.  }
9.  component FPC {
10.   attribute int value;
11.   constraints(default) {
12.     value = 1;
13.   }
14.   constraints(x1) {
15.     value = {2..4};
16.   }
17.   constraints(unknown) {
18.
19.   }
20. }
21. component BTS {
22.   attribute int fpc;
23.   constraints {
24.     FPC fpc1;
25.     fpc = fpc1.value;
26.   }
27. }
28. component Cell {
29.   attribute int neededR, realR;
30.   constraints {
31.     BTS b1;
32.     P2PMeter s[100];
33.     realR=sum([s], mRate)/P2PNo;
34.     realR>=neededR;
35.   ..
36.   }
37.   transition {
38.     forall ( P2PMeter ) {
39.       if (mdist=1 and codeset=2 )
40.         codeset.next = {2,3};
41.       if (mdist=3 and codeset=2 )
42.         codeset.next = {2,1};
43.     }
44.   }
45. }
```

Figure 3: A (partial) SIMOL program

3 SIMOL syntax and semantics

As already said, SIMOL is an object-oriented programming language. Most of the basic features have been already described elsewhere [Nica and Wotawa, 2011; 2012b]. However, in order to be self-contained, we briefly introduce and discuss SIMOL's syntax and semantics. To be more accessible for non-experts in configuration and constraint solving, we decided to adopt the syntax of Java. The program depicted in Figure 3 is a partial model used in our M2M application domain. The program comprises 4 components, which model a Point-to-Point (P2P) individually addressable smart meter, a base transceiver station (BTS), the number of serving frequencies (FPC) and a mobile cell. Every component definition starts with declaring the name of the component. Within a component, its attributes, constraints, and transitions are defined. The latter is for defining the next state in order to model discrete time and state machine models.

Syntax: Since SIMOL has a Java-like syntax, most of the defined tokens are Java-like, i.e., identifiers for any type of components and attributes, integers, and boolean literals, separators, arithmetic and relational operators ($+, -, *, /, =, <$ $, >, <=, >=, !=$), comments and also reserved keywords. In addition, it is also possible to use physical units like Watt (W), or Ampere (A), etc., for a more realistic description. Another feature of the language is that the domain of the variables values can be restricted. In Line 15 of the program depicted in Figure 3 only the values 2, 3, and 4 are allowed for variable `value`.

Every SIMOL program comprises 3 sections: (1) a knowledge based declaration section (Line 1) for organizing the files similarly to Java packages, (2) an import declaration section where knowledge bases can be loaded, and (3) component definitions (Line 2 to 45). The first 2 sections are optional, whereas the component definition section is mandatory. Each component definition starts with the keyword **component** followed by the name of the component and with an optional **extends** followed by a comma-separated list of component names. If **extends** is used, we know that the new component has one or more super components from which constraints are inherited.

In every component definition, we firstly define the component's attributes after the **attribute** keyword. For example, in Line 3 the attributes `midst`, `code set`, and `mRate` are defined. All these attributes are of type integer (**int**). Besides attributes, constraints can be defined. There are two ways of doing this. Either we use the keyword **constraints** directly followed by a block in surrounding parentheses $\{\}$, or we use the same keyword **constraints** followed by a mode name under parentheses () and again a block statement. The first definition of constraints only allows for specifying a single component behavior. The other definition makes use of modes that are needed later on for configuration. For example, in Line 11 to 19 three modes for the component `FPC` are defined. The `default` mode sets the value of variable `value` to 1. The `x1` mode restricts the domain of `value`, where the constraint solver can select one value from the range $\{2..4\}$ when computing reconfigurations. The last mode (`unknown`) does not specify any value.

In the constraint section of a component definition the following types of statements are allowed in SIMOL:

- an *empty statement*: `;`,
- a *component instance declaration*, with the possibility of initializing its attributes. See for example Line 24 of the `GPRSCell` knowledge base, where a new component

`fpc1` is generated as an instance of component `FPC`. Using this kind of statements, we define the subcomponent hierarchy in our model, i.e., the partonomy relations. The cardinality of these relations (i.e., the number of subcomponents which can be connected to a certain component) is always finite.

- an *arithmetic or/and boolean expression*
- a *conditional block* starting with the keyword **if** and optionally followed by an **else**.
- special functions like **forall**, **exist**, **sum**, **product**, **min**, and **max**, that allow to state constraints over an array of instances or values. For example, in Line 33 we sum the `mRate` attribute of all `P2PMeter` stored in variable `s`.

In addition, models written in SIMOL might be described over discrete time. For this purpose SIMOL makes use of the **transition** section. Within the transition section the new values of some, but not necessarily all variables, have to be defined. In our running example Line 37 to 43 define the next values of the `codeset` variables for all `P2PMeters`. In order to distinguish the new value of a variable in the successor state we make use of the keyword **next**. It is worth noting that in the transition section we can use all statements from the constraint section.

Semantics: The following informal definition of the semantics of SIMOL relies on mathematical equations. In particular the idea behind the semantics is to map all constraints that are assigned to one component to a set of equations. This also requires the combination of equations in case of multiple inheritance and component instances. In the first part of the definition of the semantics we ignore discrete time. We discuss this issue later in this section.

For each component C defined in SIMOL we assume a set of equations $constr_0(C)$, representing the set of constraints within the **constraints**(mode) $\{\ldots\}$ blocks. Then each constraint C_{mode} within a **constraints**($mode$) $\{\ldots\}$ block contributes to $constr_0(C)$, only if $mode$ is active. Hence, we can define this as a *conditional mode constraint* $C_{cond_{mode}}$: $if(mode\ is\ active)\ C_{mode}\ must\ be\ satisfied$ and therefore $constr_0(C)$ becomes:

$$constr_0(C) = \bigcup_{mode \in MODES(C)} constr_{mode}(C)$$

where $MODES(C)$ is the set of functional modes, defined for component C, and $constr_{mode}(C)$ is the set of *conditional mode constraints*.

Moreover, the component C also receives equations from its super components and the instances used in the component definition. Because of the possibility of having more than one instance of a component, we have to rename the variables used in the constraints of an instance. For this purpose, we assume a function $replace$ that takes constraints M and a name N and changes all variables x in M to $N.x$. Hence, the set of equations that corresponds to a particular component C is given by the following definition:

$$constr(C) = constr_0(C) \cup constr_I(C) \cup constr_V(C)$$

where $constr_I$ are the constraints inherited from the super components of C

$$constr_I(C) = \bigcup_{C' \in super(C)} constr(C')$$

$constr_V$ are the constraints coming from the components used in the definition of C (and requiring variable renaming using the function $replace$ that add a new pre-fix to the variables used in the components in order to make them unique)

$$constr_V(C) = \bigcup_{(C',N) \in vd\_inst(C)} replace(constr(C'), N)$$

Each constraint within the **constraints** $\{\ldots\}$ block contributes to $constr_0(C)$ as follows:

- C_{attr_val} : *attribute-equals-value/s* constraints, formulated with $=$ operator and applied on component attributes together with one single integer/boolean value or with a set of values;
- C_{attr_attr} : *attribute-equals-attr* constraints, formulated with $=$ operator and applied on component attributes;
- C_{num} : *numeric constraints*, formulated with basic relational operators over numeric expressions;
- C_{cond} : *conditional constraints*, $if(C_x\ is\ satisfied)\ C_y\ must\ be\ satisfied\ else\ C_z\ must\ be\ satisfied$;
- C_{exist} : *existence constraints*, $exist(at\_least(NR)\ |at\_most(NR)|NR, C, ATTR = VALUE)$, with the meaning that at most, at least or exactly NR components of a given type C have $ATTR = VALUE$.

  Note that the **forall**, **sum**, ... constraints are similarly defined.

How to handle time? Within the **transition** section we have constraints that define a relationship between the variables of a state and its successor state. In order to represent states, we introduce an index that is assigned to each variable used in $constr(C)$. Hence, what we do is to define constraints that hold in each state $i \in \{0, \ldots, s\}$, where s represents the maximum number of considered states within a reconfiguration model. These constraints are obtained from $constr(C)$ by adding an index i to the variables. We represent these constraints using the function $constr_i(C)$. For example, if `value = 1` is element of $constr(C)$, then `value_4 = 1` is element of $constr_4(C)$. Such constraints are valid within a state and therefore called *state constraints*.

In addition to state constraints we require *transition constraints*. The transition constraints can be easily computed from the **transition** section. In principle we make use of the same translation as in the **constraints** block, but also take care of the **next** attribute assigned to variables. If a variable v has such an attribute and we consider state i we replace

v.**next** with $v\_i + 1$. Variables v without the **next** attribute are changed to v_i. Hence, we obtain all transition constraints $trans_i(C)$ for state i and component C.

In summary, the constraints for the whole SIMOL program can now be obtained as follows:

$$constr = \bigcup_{i \in \{0,\ldots,s\}} \bigcup_{C} (constr_i(C) \cup trans_i(C))$$

Hence, the set of the obtained equations represents the behavior of the SIMOL program. It is worth noting that we took the semantic definition based on equations from the semantics of Modelica [Fritzson and Bunus, 2002]. In contrast to Modelica the handling of time is different as well as some of the functions that can be used within SIMOL.

4 Reconfiguration in SIMOA

Before stating a configuration algorithm, which is based on the diagnosis algorithm ConDiag [Nica and Wotawa, 2012a], we introduce and discuss some basic definitions. We first formalize the reconfiguration problem. The reconfiguration problem requires information on the current system and the new requirements. Note that the current system may fulfill the given requirements. In this case no changes of the current system are required. For the information needed of the current system we follow a component-oriented engineering approach and assume that the structure as well as the behavior has to be represented. The behavior of course has to capture those aspects relevant for configuration. In particular the functionality of the system has to be modeled.

In addition we assume that for each component of the system we know how to reconfigure the component. Here we borrow the idea coming from Model-Based Diagnosis (MBD) [Reiter, 1987; de Kleer and Williams, 1987] and introduce modes for components. Hence, every component has at least one mode. We assume the $default$ mode to be the standard mode of a component, and all other modes to be potential reconfigurations of this component. For simplicity, we introduce a function $modes : COMP \mapsto MODES$ mapping components from $COMP$ to their $MODES$. At least $default$ has to be element of $modes(c)$ for all components $c \in COMP$. The SIMOL language allows for specifying models of systems comprising components and their modes. For example, in lines 2–27 of our running example from Figure 3 the components `P2PMeter`, `FPC` and `BTS` are defined. `P2PMeter` and `BTS` only have one mode (i.e., the $default$ mode), whereas for `FPC`, 3 modes ($default$, $x1$, and $unknown$) are defined.

Besides the structure and behavior of the system, we have to define the new system requirements. System requirements in our context are nothing else than constraints, which a system has to fulfill. For example, we might say that the mobile phone network has to be capable of servicing 100 smart meters at once in a particular area, given the communication requirements of the smart meters. In the context of SIMOA this information again is specified using SIMOL. For example, lines 28–45 of the program from Figure 3 are for specifying exactly those requirements.

Definition 1 (Reconfiguration problem) *A reconfiguration problem can be defined as a tuple $(KB, COMP, MODES)$, where $KB = SD \cup REQ$ is the knowledge base comprising the model of the system SD and the requirements REQ, $COMP$ is a set of system components, and $MODES$ is the set of functional modes for the elements of $COMP$.*

The reconfiguration problem consists in searching for an assignment of modes to each component, such that the knowledge base together with this assignments is satisfiable.

As already mentioned, all information regarding the reconfiguration problem can be obtained from SIMOL programs. The program from Figure 3 allows us to derive the knowledge base KB, which is the set of equations $constr$ representing the semantics of the SIMOL program, the set of components $COMP = \{\texttt{P2PMeter}, \texttt{FPC}, \texttt{BTS}, \texttt{Cell}\}$, and the set of modes $MODES = \{default, x1, unknown\}$.

Definition 2 (Mode assignment) *Given a set of components $COMP$ and a set of functional modes $MODES$. A mode assignment M is a function $M : COMP \mapsto MODES$ mapping each component to one of its modes, i.e., for all $c \in COMP : M(c) \in modes(c)$.*

Having now all ingredients we are able to formally state a reconfiguration as follows:

Definition 3 (Reconfiguration) *Given a reconfiguration problem $(KB, COMP, MODES)$. A mode assignment M is a valid reconfiguration iff $KB \cup \{M(c) | c \in COMP\}$ is satisfiable.*

In reconfiguration we are interested in finding mode assignments that do not imply too many changes. Hence, we can use the number of required system changes to indicate the optimality of a reconfiguration. The number of changes necessary in a mode assignment is the number of used modes that are not equivalent to the $default$ mode.

Definition 4 (Number of changes) *Given a reconfiguration M for a reconfiguration problem $(KB, COMP, MODES)$. The number of changes (NOC) of M is equivalent to the number of modes in M deviating from the $default$ modes, i.e., $NOC(M) = |\{M(c) | c \in COMP \wedge M \neq default\}|$.*

We say that a reconfiguration M is optimal with respect to its NOC if it is minimal, i.e., there exists no other reconfiguration M' with $NOC(M') < NOC(M)$. This definition of minimality corresponds to cardinality minimality in diagnosis, which is different from the usually used subset minimality of diagnosis (see [Reiter, 1987]). However, for the purpose of reconfiguration minimality based on cardinality seems to be a better choice.

After stating the underlying definitions we introduce an algorithm for reconfiguration that is based on ConDiag [Nica and Wotawa, 2012a]. Computing reconfigurations in our context is nothing else than searching for minimal mode assignments, i.e., mode assignments that are as close to the original assignments as possible. When assuming that small changes lead to a satisfiable knowledge base, it would be good to start search considering small deviations of mode assignments from the $default$ mode first. The number of changes

can be increased if no solution is found. Therefore, an iterative algorithm seems to be sufficient.

Algorithm 1 reconfig$(KB, COMP, MODES, n)$

Input: A reconfiguration problem $(KB, COMP, MODES)$ and the maximum NOC n
Output: All minimal reconfigurations (up to the predefined cardinality n)

1: **for** $i = 0$ **to** n **do**
2: $CM = \{|\{M(c)|c \in COMP \wedge M \neq default\}| = i\} \cup KB$
3: $S = \mathcal{P}(\textbf{CSolver}(CM))$
4: **if** $S \neq \emptyset$ **then**
5: **return** S
6: **end if**
7: **end for**
8: **return** $\emptyset$

Algorithm 1 **reconfig** takes a reconfiguration problem and a maximum number of changes and computes all minimal reconfigurations. Algorithm 1 is an iterative algorithm that starts with no changes of modes and continues search if necessary up to the predefined value n. The termination criteria before reaching n is given in Line 4, where an 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). The **CSolver** is a constraint solver taking a set of constraints CM and is expected to return a set of mode assignments if a satisfiable solution can be found. Otherwise, the empty set is returned indicating that no reconfiguration of the given size is possible. The function $\mathcal{P}$ is assumed to map the output of the solver to a set of solutions.

In the SIMOA prototype implementation we make use of the MINION [Jefferson *et al.*, 2012; Gent *et al.*, 2006] constraint solver for this purpose, but every other constraint solver would also be sufficient providing that it is capable of handling the constraints stored in CM. Line 2 of Algorithm 1 adds a new constraint to the model stating that we are interested in finding solutions that comprise exactly i modes that are not equivalent to $default$.

Algorithm 1 obviously terminates assuming that **CSolver** terminates. The complexity is of $O(n \cdot k)$ where k is the complexity of **CSolver**. In the worst case searching for solutions for a finite constraint satisfaction problem is exponential in the size of the problem. Therefore, **reconfig** is also exponential in the worst case. However, in practice solutions can be found in a much faster way. See for example the results reported in [Nica and Wotawa, 2012a] and more recently [Nica *et al.*, 2013]. In these paper search for solutions up to a size of 3 is within seconds even for constraint satisfaction problems comprising up to 3,800 constraints. Although these results are for diagnosis, they also can be applied to configuration because of the similarity of the algorithms.

5 Empirical results

In this section we report on first empirical results obtained using a SIMOL model of our application domain, i.e., smart metering. The SIMOL source code has 95 lines of code, describing a model with one, two, or three cells, where each cell contains from 7 up to 100 `P2PMeter` components. When compiling the SIMOL program to its MINION representation, considering at maximum 5 states, we obtain up to 2,387 variables and 7,320 constraints, depending on the the number of smart meters considered. In principle, there are many possibilities of mapping SIMOL to MINION and also for computing solutions for a given maximum number of changes NOC. In the following, we discuss the encoding of SIMOL modes within MINION and show that the choice of certain MINION parameters influence the computation of reconfigurations substantially.

The mode encoding in MINION is rather straightforward. In principle, a mode of a component can be either active or inactive. Therefore, we map each mode $mode_x$ to a Boolean variable in MINION, which is 1 (true) if the corresponding component is in mode $mode_x$, or 0 (false) otherwise. In order to compute a solution for a particular NOC we have somehow to maximize the number of $default$ modes in the solution. In the first version of our implementation we used the $MAXIMISING$ option of MINION for this purpose. In addition, we decided to control the way the solver searches for a solution also by directly specifying the instantiation order for the MINION variables representing a mode. Hence, we used the MINION variable ordering ($VARORDER$) as well as the corresponding value ordering ($VALORDER$) with the following settings: for all the $default$ mode variables their values should be searched in descending order, whereas for the other mode variables the searching should be done in ascending order. The intuition behind is to prefer solutions with more $default$ modes to be true over the other solutions.

For the experiments we made use of a notebook with Intel(R) Core(TM) i7 CPU 1.73 GHz and 4 GB of RAM running under Windows 7. We obtained the results presented in the upper diagram of Figure 4 for models containing a rather small number of `P2PMeters` ranging from 7 to 50. It is worth noting that when using the $MAXIMISING$ function the measured running times exceeded 300 seconds for more than 100 meters (which is unacceptable in some situations). Hence, we decided to use only the $VARORDER$ and $VALORDER$ and ignore the $MAXIMISING$ function. From the results depicted in the bottom diagram of Figure 4 we see a substantial improvement in the measured running time. Note that the obtained results without $MAXIMISING$ were always correct.

From the diagram at the bottom of Figure 4 we can extract two observations. First, when checking only that a given system fulfills the requirements, the running time even in case of 100 `P2PMeters` is within seconds. Second, even for a NOC of size 6 the reconfiguration time never exceeds 25 seconds. Since, for the application domain these running time results are sufficient and the proposed approach is feasible.

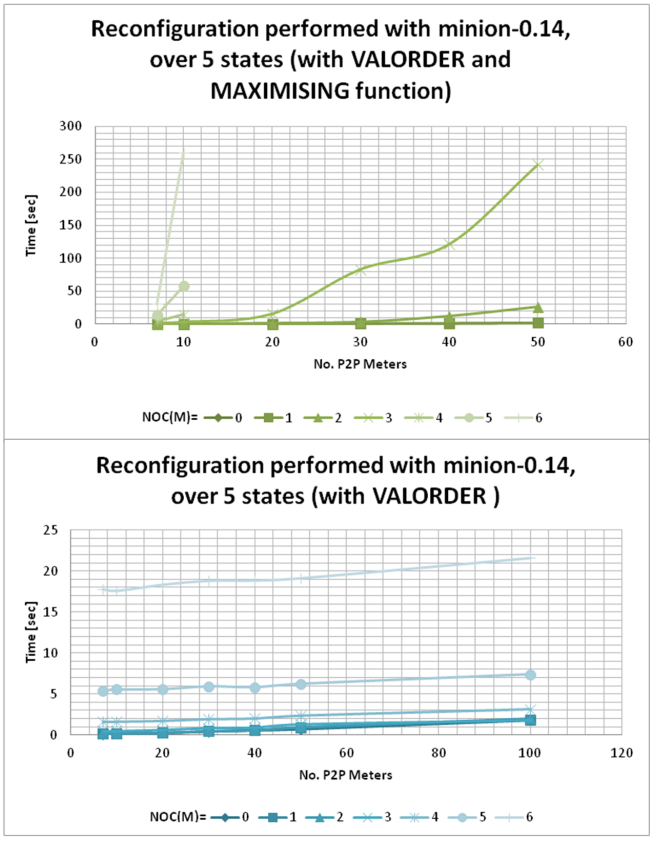


Figure 4: Comparing running times for reconfiguration, when the Integer variable domain is fixed to [0..20,000] and the number of solutions is limited to 1.

6 Related research

The idea of using constraint solvers for configuration is not new. In [Haselböck and Stumptner, 1991] the authors formalize the configuration and design problem as constraint satisfaction problem. Similarly in [Stumptner and Wotawa, 1998] the authors discuss the use of constraint solving for reconfiguration and in particular parameter reconfiguration in detail. The latter also makes use of model-based diagnosis for obtaining reconfigurations. In contrast to these previous papers our reconfiguration algorithm although relying on constraint solving is different because we compute configurations directly without making use of hitting set computation [Reiter, 1987; Greiner *et al.*, 1989] or other means for computing diagnoses [de Kleer and Williams, 1987].

The application of configuration for solving problems in the engineering domain has a long tradition. In [Stumptner *et al.*, 1994; Fleischanderl *et al.*, 1998] the authors describe the use of generative constraints for configuring large technical systems comprising thousands of components within a reasonable amount of time. Other applications include the use of configurations for web services [Felfernig *et al.*, 2002], technical products [John and Geske, 1999; John, 2000], and even telecom systems [Emde *et al.*, 1996]. Haag [Haag, 2010] discussed experiences obtained from product configuration. Although, configuration of technical products from various domains is more or less a well developed and researched field, the application to the M2M domain that requires models from the application itself and the used communication infrastructure is to our knowledge new. Moreover, besides the logical model also spatial information has to be integrated accordingly in order to come up with a correct model. The SIMOA approach provides a good bases because it allows to specify constraints dealing with Boolean and Integer values as well as discrete time. Moreover, also arrays can be used for modeling. Extensions in the direction of handling floats or strings can be implemented but require to change the underlying reasoning engine.

There are of course many languages for simulation like Modelica [Fritzson and Bunus, 2002] or Simulink [Henson, 2005] used in industry. However, these languages are mainly optimized towards simulation and therefore can be hardly used for reconfiguration. In particular such languages do not allow under-constrained models, which are necessary for our purpose when searching for appropriate modes that do not contradict the given requirements while ensuring that the requirements can be fulfilled. There are some similarities between Modelica and SIMOL but also many differences including the tight integration of component modes and the handling of discrete time.

Our previous papers mainly deal with either the application domain [Nica *et al.*, 2012] or the SIMOL language [Nica and Wotawa, 2011; 2012b]. In contrast to the latter paper, we extend the SIMOL language using the **transition** block in order to handle discrete time in the underlying models. Moreover, we discuss the algorithm for configuration in more detail in this paper.

7 Conclusions

In this paper we discussed the underlying language, definitions, and algorithms of the SIMOA approach to reconfiguration. Although, the approach has been applied to the machine-to-machine communication domain, it is not restricted to this domain. Any reconfiguration problem that can be represented using the underlying modeling language SIMOL can also be solved using the proposed SIMOA approach. SIMOL itself is an object-oriented programming language where components can be defined. The syntax of SIMOL is close to Java. The semantics has been mainly taken from the modeling language Modelica. Within the developed SIMOA prototype SIMOL is converted in MINION constraints. Hence, MINION is used as underlying constraint solver. This again does not restrict the approach since changing constraint solvers is still possible. Only, the conversion of SIMOL has to be adapted.

Besides SIMOL we also discuss the basic definitions of reconfiguration and state an algorithm that allows to find minimal reconfigurations up to a predefined size. Size in this context is defined as number of necessary changes of the system in order to fulfill all constraints. The reconfiguration algorithm derives solutions directly from the constraints (i.e., equations coming from SIMOL). This distinguishes this approach from other similar approaches where search for valid configurations is often based on conflicts and conflict resolu-

tion. First empirical results indicate that computation is sufficiently fast and that the results are within expectations. In the future it is planned to further evaluate the approach.

Acknowledgement

The work presented in this paper has been supported by the BRIDGE research project Simulation and Configuration of Mobile networks with M2M Applications (SIMOA),which is funded by the FFG.

References

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

[Emde *et al.*, 1996] Werner Emde, Christian Beilken, Josef Bording, Wolfgang Orth, Ulrike Petersen, Jörg Rahmer, Michael Spenke, Angi Voss, Stefan Wrobel, and Schlo Birlinghoven. Configuration of Telecommunication Systems in KIKon, 1996.

[Felfernig *et al.*, 2002] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Semantic Configuration Web Services in the CAWICOMS Project. In *ISWC '02 Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 192–205, 2002.

[Fleischanderl *et al.*, 1998] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner. Configuring large systems using generative constraint satisfaction. In *IEEE Intelligent Systems & their applications*, pages 59–68, 1998.

[Fritzson and Bunus, 2002] Peter Fritzson and Peter Bunus. Modelica - a general object-oriented language for continuous and discrete-event system modeling and simulation. In *Proceedings 35th Annual Simulation Symposium*, pages 365–380, 2002.

[Gent *et al.*, 2006] I. P. Gent, C. Jefferson, and I. Miguel. MINION: A Fast, Scalable, Constraint Solver. *17th European Conference on Artificial Intelligence*, ECAI-06, 2006.

[Greiner *et al.*, 1989] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[Haag, 2010] Albert Haag. Experiences with Product Configuration? http://www.minet.uni-jena.de/dbis/lehre/ss2010/konfsem/, 2010.

[Haselböck and Stumptner, 1991] Alois Haselböck and Markus Stumptner. Configuration and design as a constraint satisfaction task. In *Artificial Intelligence in Design – Proceedings of the Workshop of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, August 1991. Also appeared as Technical Report DBAI-CSP-TR 91/1.

[Henson, 2005] William Henson. Real time Control and Custom Components in the Matlab Environment. Technical report, 2005.

[Jefferson *et al.*, 2012] Christopher Jefferson, Lars Kotthoff, Neil Moore, Peter Nightingale, Karen E. Petrie, and Andrea Rendl. TheMinion Manual, Minion Version 0.15. http://minion.sourceforge.net/, 2012.

[John and Geske, 1999] Ulrich John and Ulrich Geske. Reconfiguration of Technical Products Using ConBaCon. In *Proceedings of WS on Configuration at AAAI-99*, Orlando, 1999.

[John, 2000] Ulrich John. Solving large configuration problems efficiently by clustering the ConBaCon model. In *Proceedings of the 13th international conference on Industrial and engineering applications of artificial intelligence and expert systems: Intelligent problem solving: methodologies and approaches*. Springer-Verlag New York, Inc., 2000.

[Nica and Wotawa, 2011] Iulia D. Nica and Franz Wotawa. SiMoL- A Modeling Language for Simulation and (Re-)Configuration. In *Workshop on Configuration*, pages 40–43, 2011.

[Nica and Wotawa, 2012a] 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.

[Nica and Wotawa, 2012b] Iulia D. Nica and Franz Wotawa. The SiMoL Modeling Language for Simulation and (Re-) Configuration. In *Proc. Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, pages 661–672, 2012.

[Nica *et al.*, 2012] I. D. Nica, F. Wotawa, R. Ochenbauer, C. Schober, H. Hofbauer, and S. Boltek. Model-based simulation and configuration of mobile phone networks - the SIMOA approach. In *Proc. of the ECAI 2012 Workshop on Artificial Intelligence for Telecommunications & Sensor Networks*, pages 12–17, 2012.

[Nica *et al.*, 2013] Iulia D. Nica, Ingo Pill, and Thomas Quaritsch Franz Wotawa. The Route to Success - A Performance Comparison of Diagnosis Algorithms. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Stumptner and Wotawa, 1998] Markus Stumptner and Franz Wotawa. Model-based reconfiguration. In *Proceedings Artificial Intelligence in Design*, Lisbon, Portugal, 1998.

[Stumptner *et al.*, 1994] Markus Stumptner, Alois Haselböck, and Gerhard Friedrich. COCOS - a tool for constraint-based, dynamic configuration. In *Proceedings of the 10th IEEE Conference on AI Applications (CAIA)*, San Antonio, March 1994.

New complex product introduction by means of product configuration

Martin Bonev and **Manuel Korell** and **Lars Hvam**
Technical University of Denmark, Denmark

Abstract

Configuration systems have widely been applied to efficiently address the customization responsiveness squeeze of companies dealing with Mass Customization. Over time, several frameworks have been introduced to enable their systematic planning, analyses, development and implementation. Traditional research has thereby either focused on defining modelling techniques for the configuration model of stable products, on improved configuration algorithms, or on the impact of configurators on companies' operations. However, little attention has yet been paid how the growing need for product innovation can effectively been supported. Especially for engineering companies moving towards Mass Customization, compared to mass producers the challenges caused by the complexity of their products and by the highly uncertain markets are much higher. This study develops and validates a framework which enables the use of configuration systems along the introduction of complex products. It in particular examines (1) what are suitable development strategies for configuration systems during product innovation, (2) how product development and configuration development can be aligned and managed, and (3) how supplier integration can be achieved.

1 Introduction

1.1 Background

With mass customization (MC) companies are aiming at effectively addressing the customization-responsiveness squeeze, i.e. the necessity of offering custom tailored products at nearly mass production efficiency [Tseng *et al.*, 2001]. Since its introduction in the late 1980's [Davis, 1989], the concept has received much attention from both practitioners and scientists. General strategies and advanced IT systems, such as configuration systems (CSs), have potentially helped companies to effectively cope with global competition and increased customer demands [Salvador *et al.*, 2009].

1.2 Motivation and outline of the paper

While much of the research has yet focused on developing models and theoretical frameworks, little empirical studies have explained the effective introduction of new customized products [Slamanig *et al.*, 2011]. Notably the use of configuration systems has seldom been discussed in the context of radical innovation processes [Hara *et al.*, 2012]. Thus considering the challenges of dynamically changing markets and increasing product complexity [Blecker *et al.*, 2006], further guidance based on empirical evidence is needed. Especially for engineer-to-order (ETO) manufacturers who are moving from an individual customization to a partly MC these challenges are particularly important. Compared to mass producers, their products are typically more complex and high uncertainties of demands make planning activities more difficult [Rahim *et al.*, 2003].

The emphasis of this study is therefore to investigate how new products can be launched effectively in situations in which product complexity (internal complexity) is rather high and where only little information about the customer requirements (external complexity) exists. A particular attention is thereby paid on how CSs can support product innovations for significant product renewals.

Based on a literature study (Section 2), the paper first examines existing approaches for MC with regard to the use of CSs in the context of new product introduction. Relevant frameworks are adapted to better meet the requirements of ETO manufacturers pursuing MC strategies and product innovation with product configuration (Section 3-4). Next, the newly introduced framework is applied on an industrial case study (Section 5), where a configuration model was initially developed. The achieved findings and practical implications are eventually discussed (Section 6).

2 Literature Review

2.1 Product configuration and mass customization

Offering bespoke products to customers affects the entire product realization process starting from the order acquisition to the order fulfilment [Forza and Salvador, 2002]. According to Jiao and Tseng (2004) the impact of customization can be described with the generic domains of an

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

organization [Jiao and Tseng, 2004], where to begin with customer satisfaction can be achieved through the efficient match of the requirements to the offered solution space of product variants. Salvador et al. (2009) refer to this process as assortment matching, in which suitable software helps to link the existing solution space to customer's needs [Salvador *et al.*, 2009]. The most common software systems that enable the realization of an efficient assortment matching are configuration systems [Forza and Salvador, 2002]. Being a subtype of a knowledge-based expert systems, CSs formally represent the product knowledge relevant to the customer (product features), allowing a complete definition of possible product outcomes (customized functional features) with a minimum of entities [Hvam *et al.*, 2011].

More recently, researches have investigated the use of CSs not only as sales tools, but also in support of the entire specification process, i.e. the order acquisition and order fulfilment process [Forza and Salvador, 2002]. Helo *et al.* (2010) for instance propose a business model for the use of configuration systems throughout the entire specification process of a product [Helo *et al.*, 2010]. The authors discuss how sales configuration can first be used to translate customer needs into functional requirements of a product. In the physical domain, product configuration then matches the chosen set of functionalities into design parameters. Even though not implemented in the study, process configuration can eventually be used to select on a high level suitable production and logistic steps for the subsequent processes. Figure 1 below illustrates a generic value chain of a manufacturing company including its specification process. Depending on the scope of the project, CSs can potentially be implemented to support wholly or only partly the specification process [Hvam et al., 2008].

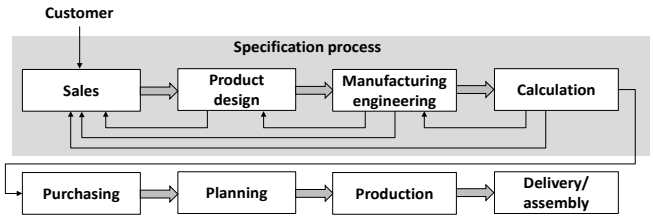


Figure 1: Generic specification process

2.2 Recent trends in product innovation

Obviously, by integrating the different customization domains into the configuration process helps to provide salesmen with more accurate estimations of time and cost of existing products. However, over time competition forces firms to update their established product portfolio. Smith et al. (2012) discuss two major reasons for companies to regularly work on product innovation:

1. customers change requirements, and
2. product performance needs to be constantly improved [Smith *et al.*, 2012].

Hence, in the first case new products are only introduced when considerable large discrepancy exists between customer needs and the provided functionality of existing products. In the latter case new ideas and technologies keep customers engaged with the products and thus stimulate sales [Howard *et al.*, 2011].

In majority of the cases, working on product innovation is typically based on existing products, where often more than 70% of the development tasks are related to redesigning, improving, and extending the products offered to the market [Ullman, 1997]. To achieve high productivity in the innovation, companies are on the one hand pressured to employ adequate tools and methods that allow an in-depth understanding and managing of knowledge related to products, processes, as well as to their project environment [Vezzetti *et al.*, 2011]. On the other hand, to compete on dynamically changing markets, it has become essential to transform the innovation process from a linear to a spiral model with short and direct iterative loops and feedback cycles [Cooper and Edgett, 2008]. By doing so, initial ideas and prototypes are immediately tested, where early feedback is used for further development [Salvador *et al.*, 2009].

As technology is progressing and being used in more and more areas of business, recent studies demonstrate that a high level of technical assessment in innovation significantly improves companies' business performance. With the use of advanced technologies, probable solutions, risks and potentials can initially be evaluated. Moreover, when considering the costs and benefits from suitable technology in early stages of the innovation process, the need for technology alliances can upfront be detected [Cooper and Edgett, 2008].

2.3 Product configuration, innovation and vendor collaboration

Despite configuration systems are playing an essential part in the customization process of manufacturers, in academia their use has typically been limited to streamline specification processes of matured and well established products, usually offered by one vendor [Blecker *et al.*, 2006; Hvam *et al.*, 2008; Forza and Salvador, 2008]. Forza and Salvador (2002) for example discuss the use of a configuration system in support of the order acquisition and fulfillment process of products from one vendor with high but relatively simple product variety [Forza and Salvador, 2002]. Hvam et al. (2006) argue for the use of configuration systems as a way to improve the quotation process of ETO products or even systems. By calculating budget quotations, the configuration system manages to create sufficiently precise price estimations offered by one company [Hvam *et al.*, 2006]. Also Haug et al. (2012) investigate the use of CSs in several manufacturers of rather complex and engineering intensive products. The authors illustrate the employment of different CS development strategies in support of specifying the existing product portfolios [Haug *et al.*, 2012].

Wang et al. (2009) introduce a framework for assessing configuration changes of exiting products. Based on the operational performance of suppliers, a generic algorithm is used to calculate how a changed part affects the preference for individual suppliers. The framework is exemplary tested on a simple electronic device. Even though the authors in-

clude the collaboration of several vendors into their framework, stable products with only minor product changes (different product variants) for relatively simple products have been examined [Wang *et al.*, 2006]. Ardissono et al. (2003) propose a theoretical framework for the use of a web-based configuration system which strives to enable the collaboration between different vendors. The authors however omit to explain how the CSs should be used in praxis, especially with regard to complex products and radical innovation [Ardissono *et al.*, 2003].

3 Research Design and Objectives

From reviewing the literature it can be stated that none of the mentioned case studies considers how CSs can be used in the cause of innovation and evolvement of a complex product family, in particular not together with the coordination between different suppliers or vendors. At the same time, prevailing on increasingly competitive markets requires efficient innovation processes which are flexible enough to quickly adapt to a fast changing environment [Cooper and Edgett, 2008]. This study therefore aims at developing a framework which addresses the dilemma of being innovative on dynamically changing markets and yet still efficiently providing custom tailored products. In order achieve practical validity, a case study with a company is performed. The collaboration is organized through action research where the researchers were actively involved in a transformation process [Coughlan and Coghlan, 2004]. The industrial partner is a start-up company, a contractor with a strategic collaboration with several ETO companies.

Already at an early stage of its establishment, the company has realized the potential of using advanced IT technologies and a well thought marketing approach to gain a competitive advantage within its industry. The alliance with the strategic partners enabled sharing the otherwise unreasonable IT investment and the related financial risks. At the same time, such a strong collaboration facilitated the exchange of knowledge concerning the products and potential market segments. Rigor of data was insured through foregoing interviews and through a series of short action research cycles conducted in the cause of twelve months.

4 A Procedure for Implementing Complex Product Configuration in NPD

Several frameworks for the development and implementation of CSs exist in literature. For the study at hand, a widely used and well-structured seven phase procedure introduced by Hvam et al. (2008) was chosen. The procedure is based on the object oriented project life cycle (analysis, design, implementation and maintenance), and further contains methods for analyzing product ranges as well as the related business processes [Hvam *et al.*, 2008]. Rather than describing each of the phases in detail, in the following, we focus our attention only on the aspects that are critical with respect to innovation and new product development (NPD).

4.1 Clarifying the innovation strategy

By implementing CS several benefits can clearly be gained [Bonev and Hvam, 2012]. Yet, when planning and performing configuration projects with complex products and multiple users, the desired results are often not being achieved. According to Haug et al. (2012) a major challenge for the success of a configuration project is that for complex products, the configuration task is difficult to be estimated. In result projects often become significantly more costly than anticipated or companies fail to create prototypes that indicate the potential benefits. Another reason for abandoning initiated configuration projects is that by implementing a CS a substantial part of the business processes have to be redesigned. In case the required organizational changes are not widely accepted by the employees, the system will most likely not be used [Haug *et al.*, 2012]. To overcome these challenges it is important to establish a clear innovation strategy that promotes configuration projects which are likely to succeed and where the risk for failure is kept to a minimum. Thus, to be able to make reasonable decisions about the right innovation strategy it is inevitable to make use of relevant performance metrics. A way of assessing the performance of NPD is through monitoring the NPD productivity measured as the output from the NPD process divided by the input [Coorper and Edgett, 2009]:

$$NPD\ Productivity = \frac{Sales\ (or\ Profit)\ from\ NPD}{R\&D\ Spending}$$

As indicated in Figure 2 below, in today's quick changing business environment the outcome of the NPD can be rather uncertain. Estimations about long term sales development of new products remain vague and can cause high risks with regard to their success on the market [Oriani and Sobrero, 2008].

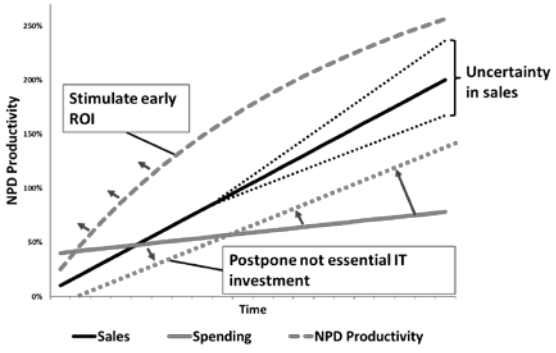


Figure 2: Effect of sales and spending on NPD productivity

In order to increase the NPD productivity and reduce risk of failure in the more reliable planning horizon, i.e. at an early stage of the innovation process, early R&D spending should be kept low. For ETO firms moving towards MC this can be achieved in two major ways. First, it is beneficial to establish strategic alliances with reliable suppliers. By sharing and coordinating innovation activities for complex products and knowledge about customer preferences and trends, individual investments and risks concerning the success on the market can be reduced [Pullen *et al.*, 2012].

Secondly, for configuration projects the R&D spending is mainly driven by the development of the configuration model and by the related IT investment. At an early stage of the configuration project it is therefore important to be clear about what are the essential ("need-to-have") functionalities the CS needs to have and which of the possible functionalities can be categorized as "nice-to-have". As the product is maturing over time and turnover from sales is increasing, further investment towards the less prioritized functionalities can be taken and the use of the CS can gradually be extended. From a financial perspective a strategic alliance and a stepwise configuration development stimulates an early return on investment (ROI) and increases the probability for more successful new product launches. Furthermore, a stepwise CS implementation encourages employees to embrace the organizational changes caused by the system, while its functionalities are being extended over time.
In sum, by involving the strategic partners in the configuration project, investment and risks can be shared and a wider range of the specification activities can be considered. Having set the requirements for the innovation strategy, in the following steps the some essential characteristics of the project life cycle will be discussed.

4.2 Developing the specification process

Before starting with a detailed analysis on the planned product innovation, if it hasn't been done yet, it is first useful to establish an overview over the current specification process at hand. From a supply chain perspective it is important to understand how the communication between various stakeholders is organized and to what extend they are influenced by the specification process. A typical sales and delivery process of ETO firms is illustrated in Figure 3 [Brunoe and Nielsen, 2012]. In contrast to mass producers, at the point of sales ETO firms usually have only a limited amount of information specifying the product and a significant amount of it has yet to be designed [Rahim and Baksh, 2003]. At the same time ETO firms still need to be able to create legally binding sales quotes which define the product to a considerable level of detail, ensuring that the communicated price and lead time results in a satisfying profit. Since generating quotations is no guarantee for receiving an order [Kingsman and De Souza, 1997], the sales process has to be effective and very cost efficient. For companies delivering ETO products the main purpose of having a CS is therefore to automate the sales and ordering process [Haug *et al.*, 2009]. In result, this initial analysis of the involved specification activities helps to assess the requirements for the subsequent automation.

Next, a TO-BE specification process supported by a CS can be defined. Scenario 2 in Figure 3 illustrates the most widespread approach for CS [Salvador *et al.*, 2009], namely a sales configurator. In other less common situations, ETO companies might have more benefits from the implementation of a solely technical CS (Scenario 2). In such a case the system would function as a design automation system for generating technical specifications for production. Due to the involvement of complex calculations, a major challenge is thereby to cover the entire technical specification [Elgh, 2008]. Next, the simultaneous implementation of both, a sales and a technical configurator is repressed by the remaining two scenarios. While in Scenario 3 two separate systems would cover the two aspects, Scenario 4 represents an integrated solution for the configuration. However, as the integration to other IT systems and to advanced calculation and CAD applications, such as to Mathcad and Inventor, is a major cost driver, in the first step this investment it is often unfeasible.

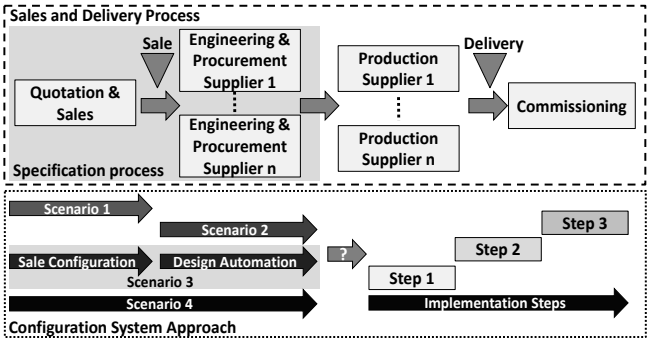


Figure 3: ETO specification and delivery process with a stepwise scenario implementation

Consequently, even though the use of advanced CS can potentially sustain the entire specification process (Scenario 4), to keep the investment costs and the organizational changes at a low level, in the first step (Step 1) of implementation, only the needed process steps are to be assisted by the system. In the subsequent steps (Step 2 etc.), more and more activities related to the specification of a product can be automated. In the majority of the cases it is feasible to start with the development of a sales CS, as for example investigated by Salvador et al. (2009). Such a system could then be used as a marketing tool, where in the introduction and growth phase of the product life cycle the focus is on creating customer awareness of the product and on trial of different product variants [Kotler *et al.*, 2012]. With the right analytical capabilities [Davenport and Harris, 2007], companies could quickly uncover customer preferences and thus further extend their product portfolio towards the required product features.

4.3 Aligning product analysis and development with configuration development

Since in most cases product innovation builds upon existing products [Smith *et al.*, 2012], after clarifying the implementation steps, an analysis of the most similar product architecture needs to be taken. Ulrich (1995) defines product architecture as: (1) the arrangement of functional elements; (2) the mapping from functional elements to physical components; and (3) the specifications of the interfaces among interacting physical components. For the analysis of the architecture, often the Quality Function Deployment (QFD) and the Design Structure Matrix (DSM) have widely been utilized. With their help customers' needs are identified and linked into the created product structure [Vezzetti *et al.*,

2011]. The employment of the Modular Function Deployment (MFD) then enables the creation on decoupled functional units, i.e. modules [Ericsson and Erixon, 1999].

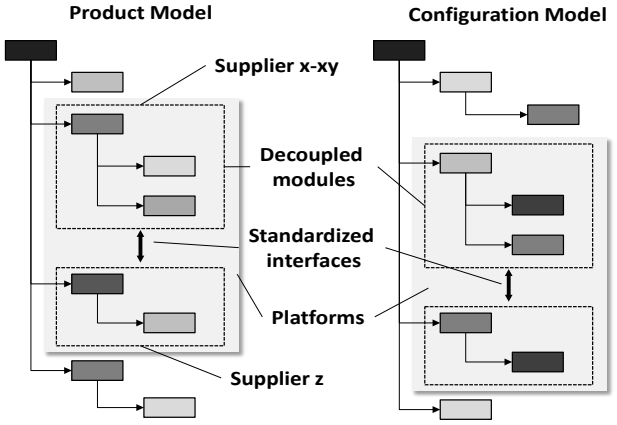


Figure 4: Aligning product model with configuration model

Another way of representing the product architecture is through the hierarchy structure of the Product Variant Master (PVM) technique. By following the basic principles of object oriented modelling, such as generalization, aggregation and association, the PVM technique uses the Unified Modeling Language (UML) standard [Hvam *et al.*, 2008]. Regardless the chosen modeling technique, with product platforms in the development process are more stable product architecture can be achieved [Meyer and Lehnerd, 1997]. To ensure the collaboration between suppliers of a complex product, the individual components should be integrated as separate modules with decoupled functionalities and with clear interfaces to the related product components. Figure 4 illustrates the integration of components coming from different vendors into the entire product model. While some of the modules may be delivered from different suppliers (indicated by "x-xy" in the figure), for other modules only one supplier ("Supplier z") may exist.

A product model generally aims at representing the physical components and their functionalities. From an object oriented perspective, the development of a configuration model however characterizes the logical combination of classes and their attributes. Each class may represent physical components or other important product characteristics. Such characteristics could e.g. describe geographical, geometrical and functional product aspects, such as the targeted market or the shape and style of a product. Depending on the modelling environment of the CS, as indicated in Figure 4 the configuration model can then be illustrated as a PVM.

Even though the composition of the configuration model might be slightly different from the one of the product model, the same structural concerns are relevant for its knowledge base. Thus, since a growing product complexity typically leads to an increasing configuration complexity, wherever possible the configuration structure should consist of separate configuration modules (classes) with encapsulated constraints [Tiihonen *et al.*, 1996]. To simplify the model, also here standard interfaces among modules with a minimum number of cross related constraints are beneficial. Classes which can be carried over across product families are then to be grouped to platforms.

Furthermore, in cases where the final product components are unclear yet, a Concurrent Engineering like approach can be achieved by the use of a "black-box" configuration [Whitney, 1988]. In this case configuration classes which contain dummy attributes and constrains for the presumed product functionalities can be established in parallel to the development of the physical product components. Once the final components and the corresponding supplier specifications are available, the placeholders created in the CS can be fed with the actual information. Finally, by using the spiral model [Cooper and Edgett, 2008; Hvam *et al.*, 2008], a quick trial and error testing of the CS helps to detect critical configuration aspects and product components for which the product information is yet fragmented or not available.

5 Applying the Framework

The described framework for using CSs in the process of NPD of complex ETO products was tested for validation on an industrial case study. The thereby gained results will in the following be briefly discussed.

5.1 Developing the TO-BE specification process at the case company

Having established and overview of the AS-IS specification process, a TO-BE specification process for a stepwise CS implementation was created. The main requirements for Step 1 were:

1. The specification errors, long lead times and limited product representation should be improved by the use of a sales configurator.
2. The sales configurator should:
    a. Contain only product features which are essential for the customer.
    b. Store not essential product features as predefined default values and represent for the majority of the cases a well-designed product [Mandl et al. 2011].
    c. Be available locally on salesmen's computers.
    d. Provide a sufficiently accurate (95%) price and lead (delivery) time estimation.
    e. Provide a 3D graphical user interface (GUI) of the product, where a direct impact of the configured commercial features on time and cost is to be seen.
    f. Generate a quotation for the customer including a description of the configured product.
    g. Save the customer's information and the configuration status for a later reconfiguration.

   h. Enable the selection of non-standard choices for better adaptation of the offered solution space.
3. The remaining specification process should be divided into a configurable technical specification process and into a non-configurable engineering and procurement process.
4. The configurable technical specification process should be supported by a technical product configurator, the remaining specifications should be created in a traditional manner (through CAD and advanced calculation systems).
5. Both, the sales and the technical CS should be based on the same configuration model.
6. The output of each of the SCs should work as input for the other SC.
7. The (technical) product configurator should:
   a. Contain all design specifications of the product which can be configured within the CS.
   b. Be available on the intranet
   c. Estimate price and lead times (production, delivery, commissioning) as accurate as possible (ca. 99%).
   d. Contain only basic descriptions and static pictures of the product.
   e. Generate technical specifications and manuals for the involved suppliers.
   f. Save the configuration status for a later reconfiguration.

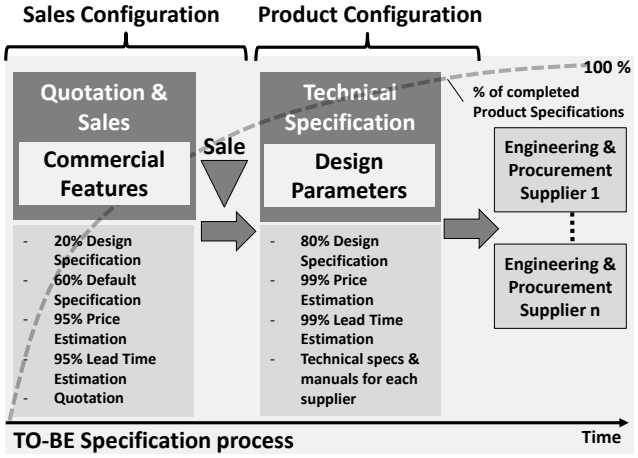


Figure 5: TO-BE Specification process of the case study

Figure 5 shows a high level representation for the chosen initial CS implementation (Step 1). To meet the requirements, a variation of Scenario 3 was selected. For the later steps of implementation (Step 2 etc.), the sales configurator should be available on the internet, where a wider range of customer awareness can be achieved. Another aspect e.g. concerns the functionalities of the technical CS. In later stages the system could have a direct integration to various CAD and calculation software, so that a higher percentage of the whole product specification can be created. However, since the product consists of components from a number of different suppliers, currently a complete definition of these 3rd party components appears to be unrealistic.

5.2 Developing the configuration model at the case company

A generic product model for yet to be developed product family was created by means of the above described modelling techniques. The corresponding configuration model was done directly in the chosen configuration software. Since both, the product and the configuration model were extended over time, the solution space of the models increased dramatically.

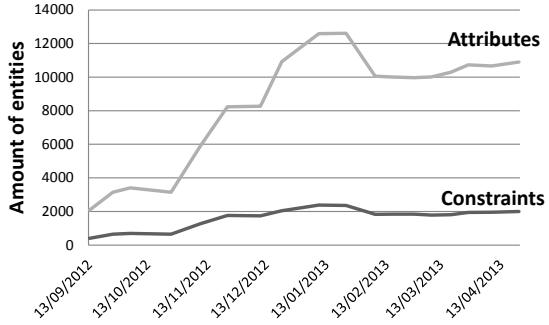


Figure 6: Progress of the configuration model

Figure 6 displays how the number of attributes and constrains of the configuration model grew as it was further completed. The growing complexity of the configuration model led to a higher computation time and to less control over the behaviour and the cause-effect relationships of the system. Hence, several initiatives were taken to reduce the structural complexity of the model. Two of them will in the following be discussed.

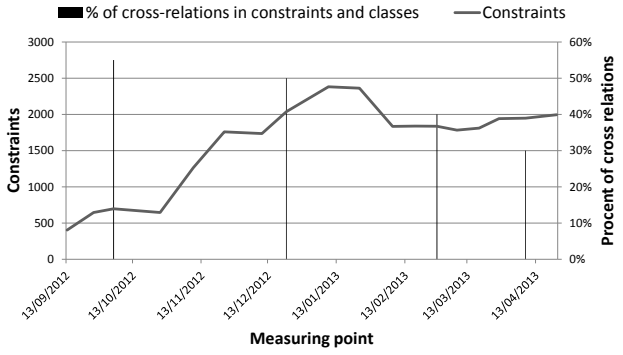


Figure 7: Reduction of cross-relations within the configuration model

To simplify the product structure, first the yet rather integrated construction of the model was redesigned to a more modular form. As described in the framework, wherever possible, it was tried utilize modularization, i.e. to make use of encapsulated classes and thus to reduce the number of cross relations. Figure 7 shows how despite a further extension of the model, a decrease from 55% to 30% cross-

relations in the model considerably reduced the number of needed constraints. Moreover, having encapsulated classes with little cross-relations provided a better overview over the entire configuration model and facilitated the inevitable debugging. In cases of unexpected behaviour, computation or even system errors, the responsible classes could easier be detected.

Solution Space of 4 related attributes for Component A and B		
Category	**Solution Space (No. of Combinations)**	**Structural Complexity**
Technically possible	19,360,000,000,000	100%
Simplified each attribute by factor 10	1,936,000,000	0.01%
Simplified each attribute by factor 100 (tolerance limit)	193,600	0.000001%

Table 1: Reduction of unnecessary attribute values

Another way to reduce the complexity of the configuration structure was to minimize ranges of attributes. Since not every technically possible attribute value is required by the customer, the characteristics of each attribute could be reduced to the tolerance limit. Table 8 exemplary depicts how a simplification of 4 attributes exponentially reduces the solution space and hence the structural complexity of the knowledge base. Instead of using the technical possible solution, by limiting the ranges with factor 100 the solution space could be reduced by factor 10^8.

6 Conclusion

When following MC principles, manufacturing companies have to consider a number of characteristics. The internal and external complexity is thereby seen as a major challenge to be handled (Blecker et al., 2006). Especially for ETO companies the movement towards MC seems to be much more complex compared to mass producers (Haug et al., 2009). Their products typically comprise a low degree of standardization with no or little commonality, their processes are seldom automated and they have little control over their customer portfolio. Our study shows that in order to better cope with arising challenges, ETO firms need to pay a particular attention on the planning phase of a new product introduction and the related product configuration development. Besides the foregoing product and process analysis (Hvam et al., 2008), several additional aspects need to be considered:

1. ETO companies using product configuration should collaborate on innovation to reduce risk and investment and to become more efficient with the new product launches.
2. Configuration systems should be planned and implemented in steps by using the spiral model, starting only from the most important "need-to-have" functionalities first.
3. Configuration systems should consider the product lifecycle objectives of products, focussing first on the creation of awareness and trial of product variants.
4. Efficiency can be gained in later steps of implementation, as functionalities are being extended, and automation and further integration to other IT systems is realized.
5. The product structure of new products needs to be redesigned in order to be configurable, while 3rd party components should preferably appear as separate modules with standardized interfaces.
6. Product model and configuration model can be created simultaneously, with a focus on stable and well known components. For yet not finally designed components dummy classes with estimated functionalities can be created.
7. In order to handle the complexity of the knowledge base, the configuration model needs to follow the same objectives as the product structure, namely; (a) the use of generic and modular yet encapsulated configuration classes with little cross related constraints (standardized interfaces), (b) the implementation of standardized and decreased attribute ranges.

References

[Adrissono *et. al.*, 2003] Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schafer, R., Zanker, M. A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems, *Ai Magazine*, Vol. 24, No. 3, pp. 93-110, 2012.

[Blecker *et al.*, 2006] Blecker, T., Friedrich, G. *Mass customization: challenges and solutions.* Springer, New York, 2006

[Bonev and Hvam, 2012] Bonev, M., Hvam, L. Analyzing the Accuracy of Calculations When Scoping Product Configuration Projects, *Lecture Notes in Computer Science*, No. 7661, pp. 331-342, 2012.

[Brunoe and Nielsen, 2012] Brunoe., D., Nielsen., P. A case of cost estimation in an engineer-to-order company moving towards mass customization, *International Journal of Mass Customisation*, Vol. 4., No. 3-4., pp. 239-254, 2012

[Cooper and Edgett, 2008] Cooper, R. G., Edgett, S. J. Maximizing productivity in product innovation, *Research Technology Management*, Vol. 51, No. 2, pp. 47-58, 2008.

[Cooper and Edgett, 2009] Cooper, R. G., Edgett, S. J. *Lean, rapid, and profitable new product development*, BookSurge, Ancaster, 2009.

[Coughlan and Coghlan, 2004] Coughlan, P., Coghlan, D. Action research for operations management, *International Journal of Operations & Production Management*, Vol. 22, No. 2, pp. 22-240, 2004.

[Davenport and Harris, 2007] Davenport, T. H., Harris, J. G. *Competing on analytics: The new science of winning*, Harvard Business School Press, Boston, 2007.

[Davis, 1989] Davis, S. M. From "future perfect": Mass customizing. *Strategy & Leadership*, Vol. 17, No. 2, pp. 16-21, 1989.

[Elgh, 2008] Elgh, F. Supporting management and maintenance of manufacturing knowledge in design automation

systems, *Advanced Engineering Informatics*, Vol. 22, No. 4, pp. 445-456, 2008.

[Ericsson and Erixon, 1999] Ericsson, A., Erixon, G. *Controlling design variants: Modular product platforms*, Society of Manufacturing Engineers, Dearborn, 1999.

[Fogliatto *et al.*, 2012] Fogliatto, F. S., da Silveira, G. J. C., Borenstein, D. The mass customization decade: An updated review of the literature, *International Journal of Production Economics*, Vol. 138, No. 1, pp. 14-25, 2012.

[Forza and Salvador, 2002] Forza, Cipriano, and Fabrizio Salvador Managing for variety in the order acquisition and fulfillment process: The contribution of product configuration systems, *International Journal of Production Economics*, Vol. 76, No. 1, pp. 87-98, 2002.

[Forza and Salvador, 2008] Forza, C., Salvador, F. Application support to product variety management, *International Journal of Production Research*, Vol. 46, No. 3, pp. 817-836, 2008.

[Hara and Arai, 2012] Hara, T., Arai, T. Encourage non-designer's design: Continuous value creation in manufacturing products and services, *Cirp Annals - Manufacturing Technology*, Vol. 61, No. 1, pp. 171-174, 2012.

[Haug *et al.*, 2009] Haug, A., Ladeby, K., Edwards, K. From engineer-to-order to mass customization, *Management Research News*, Vol. 32, No. 7, pp. 633-644, 2009.

[Haug *et al.*, 2012] Haug, A., Hvam, L., Mortensen, N. H. Definition and evaluation of product configurator development strategies, *Computers in Industry*, Vol. 63, No. 5, pp. 471-481, 2012.

[Helo *et al*., 2010] Helo, P. T., Xu, Q. L., Kyllönen, S. J., Jiao, R. J. Integrated Vehicle Configuration System - Connecting the domains of mass customization, *Computers in Industry*, Vol. 61, No. 1, pp. 44-52, 2010.

[Howard et al., 2011] Howard, T. J., Culley, S. J., Dekoninck, E. A. Reuse of ideas and concepts for creative stimuli in engineering design, Journal of Engineering Design, Vol. 22, No. 8, pp. 565-581, 2011.

[Hvam *et al.*, 2006] Hvam, L., Pape, S., Nielsen, M. K. Improving the quotation process with product configuration, Computers in Industry, Vol. 57, No. 7, pp. 607-621, 2006.

[Hvam *et al.*, 2008] Hvam, L., Mortensen, N.H., Riis, J. P*roduct Customization*, Springer, Berlin, 2008.

[Hvam *et al.*, 2011] Hvam, L., Bonev, M., Denkena, B., Schèurmeyer, J., and Dengler, B. Optimizing the order pro-cessing of customized products using product configuration, Production Engineering, Vol 5, No. 6, pp. 595-604, 2011.

[Kingsman and De Souza, 1997] Kingsman, B. G., De Souza, A. A. A knowledge-based decision support system for cost estimation and pricing decisions in versatile manufacturing companies, *International Journal of Production Economics*, Vol. 53, No. 2, pp. 119-139, 1997.

[Kotler *et al.*, 2012] Kotler, P., Keller, K. L., Brady, M. *Marketing management*, Pearson, Harlow, 2012.

[Mandl *et al.*, 2011] Mandl, M., Felfernig, A., Tiihonen, J., & Isak, K. Status Quo Bias in Configuration Systems, *Lecture Notes in Computer Science*, No. 6703, pp. 105-114, 2011.

[Meyer and Lehnerd, 1997] Meyer, M. H., Lehnerd, A. P. *The power of product platforms: Building value and cost leadership*, Free Press, New York, 1997.

[Oriani and Sobrero, 2008] Oriani, R., Sobrero, M. Uncertainty and the market valuation of R&D within a real options logic, *Strategic Management Journal*, Vol. 29, No. 4, pp. 343-361, 2008.

[Pullen *et al.*, 2012] Pullen, A., de Weerd-Nederhof, P. C., Groen, A. J., Fisscher, O. A. SME Network Characteristics vs. Product Innovativeness: How to Achieve High Innovation Performance", *Creativity and Innovation Management*, Vol. 21, No. 2, pp. 130-146, 2012.

[Rahim and Baksh, 2003] Rahim, A. R. A., Baksh, M. S. N. The need for a new product development framework for engineer-to-order products, *European Journal of Innovation Management*, Vol. 6, No. 3, pp. 182-196, 2003.

[Slamanig and Winkler, 2011] Slamanig, M., Winkler, H. An exploration of ramp-up strategies in the area of mass customisation, *International Journal of Mass Customisation*, Vol. 4, pp. 22-43, 2011.

[Smith et al., 2012] Smith, Shana, Gregory Smith, and Ying-Ting Shen, Redesign for product innovation", *Design Studies*, Vol. 33, No. 2, pp. 160-184, 2012.

Tiihonen, J., Soininen, T., Männistö, T., Sulonen, R., State-of-the-practice in product con-figuration - a survey of 10 cases in the Finnish industry. In Mäntylä, M., Finger, S., Tomiyama, T. Knowledge Intensive CAD, Vol. 1, pp. 95-114, TJ Press, Padstrow, 1996.

[Tseng and Jiao, 2001] Tseng, M., Jiao, J. *Mass Customization, Handbook of Industrial Engineering*, Wiley, New York, 2001.

[Ulman, 1997] Ullman, David G. *The mechanical design process*, McGraw-Hill, Boston, 1997.

[Ulrich, 1995] Ulrich, K. T. The Role of Product Architecture in the Manufacturing Firm, *Research Policy*, Vol. 24, No. 3, pp. 419-440, 1995.

[Vezzetti *et al.*, 2011] Vezzetti, E., Moos, S., and Kretli, S. A product lifecycle management methodology for supporting knowledge reuse in the consumer packaged goods domain, *Computer-Aided Design*, Vol. 43, No. 12, pp. 1902-1911, 2011.

[Wang *et al.*, 2009] Wang, H. S., Che, Z. H., Wang, M. J. A three-phase integrated model for product configuration change problems, *Expert Systems with Applications*, Vol. 36, No. 3, pp. 5491-5509, 2009

[Whitney, 1988] Whitney, D. E. Manufacturing by Design, *Harvard Business Review*, Vol. 07-08, pp. 83-91, 1988.

Towards Anomaly Explanation in Feature Models *

A. Felfernig[1], D. Benavides[2], J. Galindo[2], and F. Reinfrank[1]
[1]Graz University of Technology, Graz, Austria
{alexander.felfernig, florian.reinfrank}@ist.tugraz.at
[2]University of Seville, Spain
{benavides,jagalindo}@us.es

Abstract

Feature models are a wide-spread approach to variability and commonality management in software product lines. Due to the increasing size and complexity of feature models, anomalies in terms of inconsistencies and redundancies can occur which lead to increased efforts related to feature model development and maintenance. In this paper we introduce knowledge representations which serve as a basis for the explanation of anomalies in feature models. On the basis of these representations we show how explanation algorithms can be applied. The results of a performance analysis show the applicability of these algorithms for anomaly detection in feature models. We conclude the paper with a discussion of future research issues.

1 Introduction

Similar to component-oriented configuration models [Felfernig *et al.*, 2000; Felfernig, 2007], Feature Models (FM) [Kang *et al.*, 1990] are used to express variability properties of highly-variant items [Mendonca and Cowan, 2010]. Applications based on feature models help users to decide about relevant features and to learn about existing dependencies between features. Feature models can be distinguished with regard to the expressiveness of constraints defining the relationships between the different features contained in a feature model [Benavides *et al.*, 2010]. So-called *basic feature models* [Kang *et al.*, 1990] will be used as a basis for the discussions in this paper. Such models allow the definition of basic relationships between features, for example, a feature f_1 requires the inclusion of a feature f_2. *Cardinality-based feature models* [Czarnecki *et al.*, 2005] extend basic ones by also allowing cardinalities with an upper bound > 1. Finally, *extended feature models* [Batory, 2005] allow the inclusion of additional information about features in terms of feature attributes. For presentation purposes we decided to use basic feature models (see Section 2). However, the presented concepts and algorithms can be applied to advanced feature model representations as well.

Developing and maintaining large and potentially complex feature models is an error-prone activity which can be explained by the cognitive overload of software engineers and domain experts [Trinidad *et al.*, 2008; Benavides *et al.*, 2013]. In order to tackle this challenge, feature model development and maintenance processes have to be supported by intelligent techniques and tools which help to identify anomalies which become manifest in different types of inconsistencies and redundancies [Batory *et al.*, 2006; Benavides *et al.*, 2010]. An approach to the identification of *dead features* (features not part of any configuration) is presented by Trinidad et al. [Trinidad *et al.*, 2008]. The authors also introduce concepts to solve the problem of *void feature models* (no configuration exists that fulfills all the constraints in the feature model). For the identification of faulty relationships in the feature model (in these scenarios) the authors define a corresponding *diagnosis task* which is based on the concepts introduced by [Reiter, 1987]. As an alternative to the approach of [Trinidad *et al.*, 2008], White et al. [White *et al.*, 2010] show how to transform feature models into a corresponding representation of a constraint satisfaction problem (CSP) [Tsang, 1993]. On the basis of this representation, diagnoses are directly determined by the constraint solver without the support of an additional diagnostic engine. An overview of *analysis operations* (for the identification of different inconsistencies and redundancies) for feature models is provided in [Benavides *et al.*, 2010; von der Massen and Lichter, 2004].

If we are interested in *minimal* explanations (diagnoses) for feature model anomalies, the performance of the underlying algorithms becomes a challenge. An example explanation in this context would be the minimal set of constraints which have to be adapted or deleted from an inconsistent feature model (the determination of a configuration is not possible) such that the remaining constraints allow the calculation of at least one configuration. Reiter [Reiter, 1987] introduced a hitting set based approach to the determination of minimal explanations (diagnoses) – these diagnoses are also of minimal cardinality since diagnosis search is performed on the basis of breadth-first search. The idea of applying the concepts of model-based diagnosis to inconsistent constraint sets has first been introduced by Bakker et al. [Bakker *et al.*, 1993].

*This work was supported, in part, by the Austrian Research Promotion Agency under the project ICONE (827587), the European Commission (FEDER), the Spanish Government under project SETI (TIN2009-07366), and by the Andalusian Government under project THEOS (TIC-5906).

Felfernig et al. [Felfernig *et al.*, 2004] continued this work by introducing an approach to the automated testing and debugging of configuration knowledge bases where test cases are used to induce conflicts in the knowledge base. These conflicts are then resolved on the basis of the hitting set algorithm [Reiter, 1987]. First experiences from the application of these testing and debugging approaches in industrial scenarios are reported by Fleischanderl [Fleischanderl, 2002]. Junker [Junker, 2004] introduced the QuickXPlain (QX) algorithm. QX is an efficient divide-and-conquer based approach to the determination of minimal conflicts which can then be exploited for the determination of diagnoses. In this paper we show how diagnosis and redundancy detection algorithms can be applied to support feature model analysis operations [Benavides *et al.*, 2010]. In this context we show how to apply the diagnosis algorithm FASTDIAG [Felfernig *et al.*, 2012] (an algorithm with no need of determining conflict sets) and introduce the FMCORE algorithm which allows the detection of redundancies in feature models.

The work presented here is in the line of research dedicated to the development of intelligent quality assurance mechanisms for configuration knowledge bases [Felfernig *et al.*, 2004]. The major contributions of this paper are the following. First, we advance the state of the art in feature model anomaly detection by formalizing the anomaly types discussed in the feature modeling community on the basis of the concepts of inconsistency and redundancy. Second, we introduce the FMCORE algorithm for the detection of redundant constraints in feature models. Furthermore, we show how to apply the FASTDIAG algorithm [Felfernig *et al.*, 2012] for *explaining* different types of inconsistencies in feature models. All anomaly types will be discussed in detail in Section 3 in combination with corresponding explanation approaches.

The remainder of this paper is organized as follows. In Section 2 we introduce a simple feature model (operating system configuration) which will be used as working example throughout the paper. Furthermore, we introduce the definitions of a *feature model configuration task* and a corresponding *feature model configuration*. In Section 3 we introduce different relevant forms of anomalies in feature models together with their formal definitions. The corresponding anomaly detection algorithms FASTDIAG and FMCORE are explained in Section 4. The performance of these algorithms is analyzed in Section 5 on the basis of selected feature models from the *S.P.L.O.T.*[1] repository. A discussion of further research issues and a conclusion is provided in Section 6.

2 Feature models

A feature model (FM) defines a set of possible products of a domain in terms of features and the relationships between them [Wang *et al.*, 2010]. Features are arranged hierarchically (tree structure with one so-called *root feature* f_r *(*$f_r = true$*)*) [Benavides *et al.*, 2010] where the nodes are the features and the edges are relationships (constraints) [Segura *et al.*, 2010]. For a more detailed overview of different feature model representations we refer the reader to [Batory, 2005; Benavides *et al.*, 2010].

[1] See www.splot-research.org.

Semantics of Feature Models. Our representation of FMs is based on the notation introduced in [Benavides *et al.*, 2010]. Relationships (constraints) in FMs are represented in terms of six different types of constraints [Batory, 2005; Benavides *et al.*, 2010; Segura *et al.*, 2010]: *mandatory*, *optional*, *alternative*, *or*, *requires*, and *excludes*. FMs are representing configurable products which can be formalized in the form of a constraint satisfaction problem (CSP) [Tsang, 1993] where each variable f_i has the assigned domain $d_i = \{true, false\}$. We define a *feature model configuration task* as follows (see Definition 1).

Definition 1 (FM Configuration Task). A feature model (FM) configuration task is defined by the triple (F,D,C) where $F = \{f_1, f_2, ..., f_n\}$ is a set of features f_i, $D = \{dom(f_1), dom(f_2), ..., dom(f_n)\}$ ($dom(f_i) = \{true, false\}$) is the set of corresponding feature domains, and $C = CR \cup CF$ is a set of constraints restricting the possible configurations which can be derived from the feature model. In this context, $CR = \{c_1, c_2, ..., c_k\}$ represents a set of requirements (of a specific user) and $CF = \{c_{k+1}, c_{k+2}, ..., c_m\}$ a set of feature model constraints.

On the basis of this definition of an feature model configuration task, we now introduce the definition of a *configuration* for a feature model (FM) configuration task (Definition 2).

Definition 2 (FM Configuration). A feature model (FM) configuration for a given FM configuration task is a *complete* assignment of the variables $f_i \in F$. Such a configuration is *consistent* iff the constraints $c_i \in C$ are not contradicting with the variable assignment. Furthermore, an FM configuration is *valid*, if it is consistent and complete.

Feature Model Constraint Types. Six basic types of constraints can be included in CF [Benavides *et al.*, 2010]. These constraint types are the following – their representation in a graphical feature model is shown in the example of Figure 1. In the following we introduce the semantics of these six types of constraints – this semantics is based on the definition given in [Benavides *et al.*, 2010].

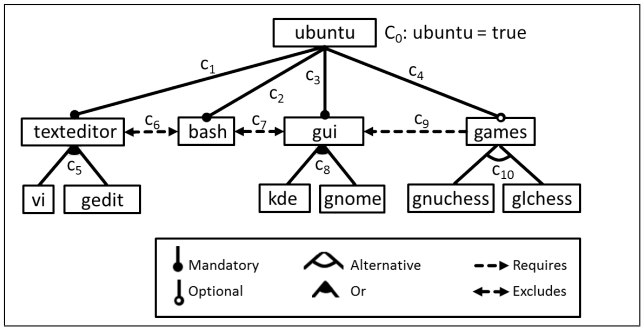


Figure 1: Feature model (FM) with faulty model elements.

Mandatory: a feature $f_2 \in F$ is *mandatory* if it is in a mandatory relationship with another feature $f_1 \in F$. This means, if f_1 is part of the configuration, f_2 must be part of the configuration as well (and vice-versa). The formalization of this constraint type (relationship) is realized on the basis of an equivalence: $f_1 \leftrightarrow f_2$. In Figure 1 the feature *gui* is a mandatory feature connected to the feature *ubuntu*.

Optional: a feature $f_2 \in F$ can (but must not) be included in the configuration in the case that feature $f_1 \in F$ is part of the configuration. This type of constraint can be formalized on the basis of an implication: $f_2 \rightarrow f_1$. In Figure 1 the feature *games* is an optional feature connected to *ubuntu*.

Alternative: only one feature $f_b \in F = \{f_1, f_2, ..., f_k\}$ can be selected if feature f_a is selected. The property can be formalized as follows: $f_1 = true \leftrightarrow (f_2 = false \wedge ... \wedge f_k = false \wedge f_a = true) \wedge ... \wedge f_k = true \leftrightarrow (f_1 = false \wedge ... \wedge f_k - 1 = false \wedge f_a = true)$. In Figure 1 an example of a feature f_a is *games*, the subfeatures are *gnuchess* and *glchess*.

Or: at least one feature $f_b \in F = \{f_1, f_2, ..., f_k\}$ must be part of the configuration if feature f_a is part of the configuration. This property can be formally defined with $f_a \leftrightarrow \{f_1, f_2, ..., f_k\}$. In Figure 1 an example of a feature f_a is *gui*, the subfeatures are *kde* and *gnome*.

Requires: a feature f_2 must be included in a configuration if feature f_1 is included. This requires relationship can be defined with $f_1 \rightarrow f_2$. In Figure 1 an example of a *requires* relationship is *games* $\rightarrow$ *gui*.

Excludes: it is not allowed to combine two features f_1 and f_2 in the same configuration, i.e., feature f_1 excludes feature f_2 and vice versa: $\neg(f_1 \wedge f_2)$. In Figure 1 an example of an *excludes* relationship is $\neg(bash \wedge gui)$. Note that this is a possible faulty constraint to be detected by diagnosis.

Requires and *excludes* constraints are also denoted as *cross-tree constraints*. Finally, the set CR (customer requirements) is an additional set of constraints to be taken into account when determining configurations (solutions). The set CR specifies a set of key features which have to be included in the FM configuration for a specific user (customer).

Example Feature Model. A simple example feature model (from the domain of *operating systems*) is depicted in Figure 1. This model specifies a set of features relevant for configuring an *ubuntu* operating system installation together with constraints between the features. Note that *faulty elements* (constraints) are contained in this model – our goal in the remainder of this paper will be to introduce algorithms which help to identify and explain such faulty constraints.

The CSP-based representation [Tsang, 1993] of the feature model shown in Figure 1 is the following - a representation as FM configuration task = (F,D,C=$CR \cup CF$).

- $F = \{ubuntu, texteditor, bash, gui, games, gedit, vi, kde, gnome, gnuchess, glchess\}$
- $D = \{dom(ubuntu) = \{true, false\}, dom(texteditor) = \{true, false\}, dom(bash) = \{true, false\}, dom(gui) = \{true, false\}, dom(games) = \{true, false\}, dom(gedit) = \{true, false\}, dom(vi) = \{true, false\}, dom(kde) = \{true, false\}, dom(gnome) = \{true, false\}, dom(gnuchess) = \{true, false\}, dom(glchess) = \{true, false\}$
- $CR = \{c_0: ubuntu = true\}$
- $CF = \{ c_1 : ubuntu \leftrightarrow texteditor, c_2 : ubuntu \leftrightarrow bash, c_3: ubuntu \leftrightarrow gui, c_4: games \rightarrow ubuntu, c_5: texteditor \leftrightarrow gedit \vee vi, c_6: \neg texteditor \vee \neg bash, c_7: \neg bash \vee \neg gui, c_8: gui \leftrightarrow kde \vee gnome, c_9: games \rightarrow gui, c_{10}: (gnuchess \leftrightarrow \neg glchess \wedge games) \wedge (glchess \leftrightarrow \neg gnuchess \wedge games)\}$

3 Anomaly Patterns in Feature Models

Anomalies can be defined as patterns in data that do not conform to a well defined notion of normal behavior [Chandola *et al.*, 2009]. Trinidad et al. [Trinidad *et al.*, 2008] are using the term *error* for *incorrect definitions of relationships*, i.e., *the set of products described by a feature model does not match the SPL (software product line) it describes*. We interpret *anomalies* in the sense of [Trinidad *et al.*, 2008]: undesirable FM properties in terms of different facets of contradictory and redundant information contained in the FM.

Handling Inconsistencies. Inconsistent feature models include contradictory constraints $c_i \in C$ that can not be satisfied at the same time, leading to no valid instances derivable from FMs [Wang *et al.*, 2010]. For a given FM configuration task this means that no solution can be identified. In our working example (the FM of Figure 1) no solution can be identified due to an inconsistent constraint set C=$\{c_1, c_2, ..., c_{10}\}$.[2] Inconsistent sets of constraints can be defined on the basis of the concept of conflict sets [Junker, 2004] (see Definition 3).

Definition 3 (Conflict Set) A conflict set $CS \subseteq C$ is a set of constraints s.t. CS is inconsistent. CS is minimal iff there does not exist a conflict set CS' with ($CS' \subset CS$).

Based on Definition 3, we can identify minimal sets of constraints $CS_i \subseteq C$, such that CS_i is inconsistent. As long as there are conflicts in a given constraint set of a feature model, no solutions for the underlying FM configuration task can be identified. Our example feature model (see Figure 1) includes two minimal conflict sets which are $CS_1 = \{c_1, c_2, c_6\}$ and $CS_2 = \{c_2, c_3, c_7\}$. Each of these sets is a minimal set such that (1) no solution (configuration) can be identified and (2) none of the subsets of CS_i is inconsistent. As a consequence (due to their minimality property) conflicts (represented by conflict sets) can be resolved by simply deleting one constraint from the set.

The resolution of all conflicts (represented by conflict sets) can be based on the determination of the corresponding hitting sets (also denoted as diagnoses [Reiter, 1987]). The problem of identifying minimal sets of constraints which have to be adapted or deleted from the feature model such that the remaining constraints become consistent can be represented as an FM diagnosis task (see Definition 4).

Definition 4 (FM Diagnosis Task) A feature model diagnosis task (FM diagnosis task) is a tuple (S, AC) where $S \subseteq AC$ are constraints of the feature model. The task is to identify a minimal set of constraints which have to be deleted from S s.t. consistency can be restored in the feature model.

In this context, S helps us to focus our diagnostic activities, i.e., to focus on those model parts where we suspect faulty constraints. If no such suspects exist, S can be set to AC. An FM diagnosis, i.e., a solution to an FM diagnosis task can be defined as follows (see Definition 5).

Definition 5 (FM Diagnosis) A feature model diagnosis (FM diagnosis) is a set of constraints $\Delta \subseteq S$ with $AC - \Delta$ is consistent. Δ is minimal iff there does not exist a set Δ' with $\Delta' \subset \Delta$ and Δ' has the diagnosis property as well.

[2] Note that we interpret the constraint $c_0 : ubuntu = true$ as element of the (customer) requirements CR.

The diagnoses for our example FM diagnosis task are $\Delta_1 = \{c_1, c_3\}$, $\Delta_2 = \{c_1, c_7\}$, $\Delta_3 = \{c_2\}$, $\Delta_4 = \{c_3, c_6\}$, $\Delta_5 = \{c_6, c_7\}$. These represent five ways to delete (adapt) constraints from (in) the feature model such that at least one configuration can be determined. The calculation of all Δ_i is sketched in Figure 2. The underlying assumption in this example is that – conform to the algorithm introduced by Reiter [Reiter, 1987] – the search tree (hitting set directed acyclic graph – HSDAG) is expanded in breadth-first manner.

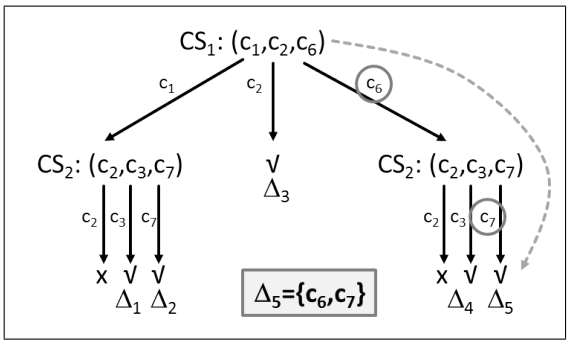


Figure 2: Determination of diagnoses for a given inconsistent feature model (FM). The following discussion of anomaly types assumes that $\Delta_5 = \{c_6, c_7\}$ was chosen.

One possible approach to determine the complete set of diagnoses is based on the *hitting set directed acyclic graph* (HSDAG) algorithm introduced by Reiter [Reiter, 1987]. The basic idea of this algorithm is to determine a conflict (in the example $CS_1 : \{c_1, c_2, c_6\}$) and then to resolve this conflict. If this conflict is resolved (e.g., by deleting the constraint c_1) the algorithm checks whether further conflicts exist in the feature model. In our example this is the case and the next determined conflict set is $CS_2 : \{c_2, c_3, c_7\}$. If we delete, for example c_3 from CS_2, we receive the diagnosis $\Delta_1 = \{c_1, c_3\}$. In a similar fashion all other diagnoses can be determined. Note that $\{c_1, c_2\}$ is not a (minimal) diagnosis since $\{c_2\}$ is already a diagnosis. The HSDAG algorithm is a traditional way of determining diagnoses – more efficient approaches will be presented in Section 4.

Feature Model Anomaly Patterns. We can now discuss in more detail different basic types of feature model *anomalies*. Ways to explain these anomalies and related algorithms will then be discussed in detail in Section 4. An overview of these anomalies and related property checks is shown in Table 1. The following types of anomalies are taken from Benavides et al. [Benavides *et al.*, 2010].

Void feature model. If model constraints in CF are inconsistent (inconsistent(CF $\cup$ c_0)), we are interested in solutions to the FM diagnosis task (S=CF, AC = CF $\cup$ c_0). In this case we want to figure out which are the minimal sets of constraints that are responsible for the given inconsistency in the feature model. We do not include c_0 (e.g., $c_0 : ubuntu = true$) in the set S since we are not interested in changing this constraint. The feature model of our example (see Figure 1) is an example of a void feature model.

Note that for the following discussions we assume that $\Delta_5 = \{c_6, c_7\}$ (see Figure 2) has been chosen by the engineer and $\{c_6, c_7\}$ have been deleted from the feature model.

Dead feature f_i. If a feature f_i is not included in any of the possible configurations (i.e., inconsistent(CF $\cup$ $f_i = true$)), we are interested in solutions to the FM diagnosis task (S = CF, AC = CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i = true\}$). This way we are able to figure out the minimal sets of constraints that are responsible for the non-acceptance of f_i. In our working example, there is no such dead feature (assuming that the constraints in Δ_5 have been deleted from the feature model). If we would substitute the constraint $c_9 : games \rightarrow gui$ with $c_9 : \neg gui \vee \neg games$, the feature *games* would be a dead feature. If we then want to make *games* a feature which is included in at least one configuration, the diagnoses for (S = CF, AC = CF $\cup$ $\{c_0\}$ $\cup$ $\{games = true\}$) are $\Delta_1 = \{c_3\}$ and $\Delta_2 = \{c_9\}$.

Conditionally dead feature f_i. Such a feature f_i is not included in all of the possible configurations, i.e., consistent (CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i$=false$\}$) and consistent (CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i$=true$\}$). If we want to have f_i in each configuration, we have to add $\{f_i = true\}$ to the set CF. In our working example, *games* is a conditionally dead feature since there are also solutions with no inclusion of this feature. In order to make *games* part of every possible feature model configuration, we have to make this clear in the feature model. One way to achieve this would be to convert constraint c_4 into a mandatory constraint – this would have the same effect as adding $games = true$ as an additional constraint to CF.

Full mandatory feature f_i. A feature f_i is fully mandatory if it is included in every possible solution (configuration), i.e., inconsistent(CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i = false\}$). If we want to adapt the feature model in such a way that it also allows f_i to be not included, we can determine the corresponding (minimal) sets of responsible constraints by solving the FM diagnosis task (S=CF, AC= CF $\cup$ $\{c_0\}$ $\cup$ $\{f_i = false\}$). In our working example, the feature *gui* is a full mandatory feature since it is part of every possible configuration. If we want to allow configurations where *gui* is not included, the only diagnosis for (S=CF, AC= CF $\cup$ $\{c_0\}$ $\cup$ $\{gui = false\}$) is $\Delta_1 = \{c_3\}$.

False optional feature f_i. A *false optional feature* f_i is included in all configurations (e.g., products of a product line) although it has not been modeled as mandatory. If we replace the constraint $c_9 : games \rightarrow gui$ with $c_9 : gui \rightarrow games$, the feature *games* becomes a false optional feature since it is included in every possible configuration. An alternative interpretation of a false optional feature focuses on the *optional* relationship between a feature f_{par} and f_{opt}. If the consistency check of (CF $\cup$ $\{c_0\}$ $\cup$ $\{f_{par} = true \wedge f_{opt} = false\}$) returns *false* (and f_{par} = $true$), the feature f_{opt} is not an option. In our example (under the assumption that c_9 is adapted as mentioned), the diagnosis for (S = CF, AC = CF $\cup$ $\{c_0\}$ $\cup$ $\{ubuntu = true \wedge games = false\}$) is $\Delta_1 = \{c_3\}$.

Redundant constraint c_i. In our working example the constraint $c_9 : games \rightarrow gui$ is redundant since *gui* is a full mandatory feature. If we check the consistency of {CF - $\{c_9\}$ $\cup$ $\neg$CF} we see that c_9 is redundant since the expression is inconsistent. In other words, CF - $\{c_9\}$ $\models$ c_9, i.e., c_9 logically follows from CF - $\{c_9\}$ – therefore it is redundant. The second redundant constraint in our working example is c_4 since the feature *ubuntu* is a full mandatory feature as well. Con-

Analysis operation	Property Check	Explanation (Diagnosis Task)
Void feature model	inconsistent(CF $\cup \{c_0\}$)?	FASTDIAG(CF,CF $\cup \{c_0\}$)
Dead (f_i)	inconsistent(CF $\cup \{c_0\} \cup \{f_i$=true$\}$)?	FASTDIAG(CF,CF $\cup \{c_0\} \cup \{f_i$ = true$\}$)
Conditionally dead (f_i)	consistent(CF $\cup \{c_0\} \cup \{f_i$=false$\}$) and consistent(CF $\cup \{c_0\} \cup \{f_i$=true$\}$)?	CF $\leftarrow$ CF $\cup \{f_i$=true$\}$
Full mandatory (f_i)	inconsistent(CF $\cup \{c_0\} \cup \{f_i$=false$\}$)?	FASTDIAG(CF,CF $\cup \{c_0\} \cup \{f_i = false\}$)
False optional (f_{opt})	inconsistent(CF $\cup \{c_0\} \cup$ $\{f_{par}$=true $\wedge f_{opt}$=false$\}$)?	FASTDIAG(CF, CF $\cup \{c_0\} \cup$ $\{f_{par} = true \wedge f_{opt} = false\}$)
Redundant (c_i)	inconsistent((CF $\cup \{c_0\}$ - $\{c_i\}$) $\cup \neg$(CF $\cup c_0$))?	$c_i \notin$ FMCORE(CF $\cup \{c_0\}$)

Table 1: Feature model analysis operations, property checks, and related explanations. For example, figuring out whether a feature model is void (no solution can be found) can be determined on the basis of a consistency check *inconsistent* (CF $\cup \{c_0\}$). A related explanation can be determined by solving the FM diagnosis task (CF, CF $\cup \{c_0\}$). The related diagnosis (FASTDIAG) and redundancy detection algorithm (FMCORE) are discussed in Section 4.

sequently, the constraints $\{c_4, c_9\}$ can be deleted from the feature model without changing the underlying semantics.[3]

In the following section we focus on the presentation of two algorithms which help to determine explanations for the different feature model anomaly patterns.

4 Explaining Anomalies

The two basic algorithms for determining diagnoses and redundancies are FASTDIAG and FMCORE. FASTDIAG [Felfernig *et al.*, 2012] is a divide-and-conquer algorithm that supports the efficient determination of minimal diagnoses without the need of having conflict sets available. FMCORE is an algorithm which focuses on the determination of *minimal cores*, i.e., redundancy-free subsets of a constraint set.

Determination of Diagnoses. In FASTDIAG (see Algorithm 1), the set S represents the set of constraints where a diagnosis should be searched, The set AC contains all constraints of the feature model. For example, if we want to diagnose a *void feature model* (CF $\cup \{c_0\}$ is inconsistent – see Table 1), we would activate the algorithm with FASTDIAG(CF,CF $\cup \{c_0\}$), i.e., S = CF and AC = CF $\cup \{c_0\}$. We do not include c_0 in the set of diagnosable constraints since c_0 (the root constraint) is assumed to be correct (e.g., c_0 : $ubuntu = true$). First, the algorithm (see Algorithm 1) checks whether the considered constraint set can be diagnosed (if the set S is empty, no diagnosis will be found) and whether the constraints in AC-S are inconsistent (in this case no diagnosis can be determined).

Algorithm 1 FASTDIAG(S, AC): Δ

if $isEmpty(S)$ *or* $inconsistent(AC - S)$ **then**
  $return\ \emptyset$;
else
  $return\ DIAG(\emptyset, S, AC)$
end if

The major idea of FASTDIAG (and its subfunction DIAG – see Algorithm 2) is to divide a set S of inconsistent constraints into two subsets S_1 and S_2. If the first part becomes consistent, the diagnosis is searched in the other part and the first part can be omitted (no constraints part of the diagnosis will be found there). If a *singleton* constraint of S triggers an inconsistency, this constraint is considered a part of the diagnosis. FASTDIAG determines exactly one diagnosis at a time. If we want to determine more than one or even the complete set of diagnoses, we need to combine FASTDIAG with a corresponding algorithm that supports the construction of HSDAGs. The discussion of this approach is outside the scope of this paper. We want to refer the reader to the work of Felfernig et al. [Felfernig *et al.*, 2012]. Compared to traditional diagnosis approaches, FASTDIAG needs in the worst case $2d \times log_2(\frac{n}{d}) + 2d$ consistency checks where d is the number of constraints in the minimal diagnosis and n is the number of constraints in S [Felfernig *et al.*, 2012]. The corresponding best case complexity in terms of the number of consistency checks is $log_2(\frac{n}{d}+2d)$. A similar worst case (and best case) complexity in traditional diagnosis approaches can be expected for each determination of a conflict set (see, e.g., Figure 2) [Felfernig *et al.*, 2012].

Algorithm 2 DIAG(D, $S = \{s_1, ..., s_r\}$, AC): Δ

if $D \neq \emptyset$ *and* $consistent(AC)$ **then**
  $return\ \emptyset$;
end if
if $singleton(S)$ **then**
  $return\ S$;
end if
$k \leftarrow \lceil \frac{r}{2} \rceil$;
$S_1 \leftarrow \{s_1, ..., s_k\}; S_2 \leftarrow \{s_{k+1}, ..., s_r\}$;
$\Delta_1 \leftarrow DIAG(S_2, S_1, AC - S_2)$;
$\Delta_2 \leftarrow DIAG(\Delta_1, S_2, AC - \Delta_1)$;
$return(\Delta_1 \cup \Delta_2)$;

Determination of Redundancies. A constraint f_i of a feature model (represented by the constraint set CF) is redundant if its deletion from the model does not change the set of possible solutions. More formally, CF - $\{f_i\} \models f_i$ which means that f_i logically follows from CF - $\{f_i\}$ and therefore is redundant. An algorithm for redundancy detection should definitely not check redundancy properties on the basis of concrete configurations since such an approach becomes com-

[3]Note that redundancies can also be intended to achieve goals such as improving understandability or increasing efficiency – a discussion of related issues is outside the scope of this paper.

Feature Model: Car Selection	#Variables: 72		#Constraints:96			
# Diagnoses	Inconsistency Rate					
	2% (8 diagnoses)		5% (64 diagnoses)		7% (182 diagnoses)	
	FASTDIAG	HSDAG	FASTDIAG	HSDAG	FASTDIAG	HSDAG
1	452	874	561	1888	858	5366
2	749	890	920	1891	1638	5382
3	1045	921	1294	2138	2059	5506
4	1373	936	1653	2143	2324	5522
5	1529	968	1872	2262	2464	5544
10	–	–	2511	2418	2932	5709
20	–	–	2964	2450	3806	6162
all	1632	1027	4383	3339	11856	8860

Table 2: Evaluation of FASTDIAG and HSDAG with the *Car Selection* feature model from S.P.L.O.T.

Feature Model: SmartHome V. 2.2	#Variables: 61		#Constraints:63			
# Diagnoses	Inconsistency Rate					
	2% (8 diagnoses)		5% (12 diagnoses)		7% (77 diagnoses)	
	FASTDIAG	HSDAG	FASTDIAG	HSDAG	FASTDIAG	HSDAG
1	297	920	312	952	577	2683
2	437	967	452	968	951	2684
3	609	983	592	983	1341	2686
4	734	998	733	1139	1762	2699
5	843	1014	842	1155	2090	2671
10	–	–	967	1529	2792	2715
20	–	–	–	–	3369	2746
all	1155	1061	1606	1545	6224	3151

Table 3: Evaluation of FASTDIAG and HSDAG with the *SmartHome V 2.2* feature model from S.P.L.O.T.

pletely inefficient even in the case of simple feature models. The basic idea of the FMCORE algorithm is to iterate over the given set of constraints (S) and for each constraint $c_i \in S$ to check whether the deletion of c_i changes the semantics of S. The assumption is that if c_i is non-redundant, its deletion from S will change the semantics of S, i.e., $S - \{c_i\} \cup \overline{S}$ becomes consistent. All these individual redundant constraints are deleted from S_{temp} (a temporal copy of S). Finally, the algorithm returns the set S_{temp} which represents a minimal core, i.e., the original set S without redundant constraints.

Note that – instead of checking the inconsistency of $C_S - \{c_i\} \cup \overline{S}$ (see, e.g., [Felfernig *et al.*, 2011]) – FMCORE systematically reduces the number of constraints to be checked in $\overline{S}$. Given a configuration knowledge base S and its complement $\overline{S}$, the (in)consistency check of $S - \{c_i\} \cup \overline{S}$ can be reduced to the inconsistency check of $S - \{c_i\} \cup \overline{S'}$ where $\overline{S'} = \{\neg c_i\}$. If we assume that $S = \{c_1 \wedge c_2 \wedge .. \wedge c_m \wedge c_{m+1} \wedge .. \wedge c_n\}$, $\overline{S} = \{\neg c_1 \vee \neg c_2 \vee .. \vee \neg c_m \vee \neg c_{m+1} \vee .. \vee \neg c_n\}$, and $\gamma = \{c_{m+1} \wedge .. \wedge c_n\}$ then the consistency check of $S - \gamma \cup \overline{S}$ can be reduced to $\{c_1 \wedge c_2 \wedge .. \wedge c_m\} \cup \{\neg c_{m+1} \vee .. \vee \neg c_n\}$. In FMCORE (Algorithm 3) this property is taken into account.

The number of consistency checks of FMCORE in the best case equals the number of consistency checks in the worst case – in both cases the number of consistency checks needed is exactly n (the number of constraints in S).

In order to analyze the performance of FASTDIAG and FMCORE we conducted a performance analysis for both algorithms on the basis of different feature models provided by the *S.P.L.O.T.* repository. The results of this analysis are presented in the following section.

Algorithm 3 FMCORE(S): Δ

```
{S: the (redundant constraint set)}
{S̄: the complement of S}
{Δ: set of redundant constraints}
S_temp ← S;
for all c_i in S_temp do
  if isInconsistent((S_temp − {c_i}) ∪ {¬c_j}) then
    S_temp ← S_temp − {c_i};
  end if
end for
return S_temp;
```

5 Performance Evaluation

For evaluation purposes we selected different feature models offered by the *S.P.L.O.T.* repository: *Car Selection* (Table 2), *SmartHome V. 2.2.* (Table 3), and *Xerox* (Table 4). In order to evaluate the performance of FASTDIAG, we randomly inserted additional cross-tree constraints in the feature models for inducing inconsistencies which could then be exploited for determining minimal diagnoses. For a systematic evaluation we generated different versions of the (inconsistent) feature models which differed in terms of their inconsistency

Feature Model: Xerox	#Variables: 172		#Constraints:205			
# Diagnoses	Inconsistency Rate					
	2% (140 diagnoses)		5% (84 diagnoses)		7% (55 diagnoses)	
	FASTDIAG	HSDAG	FASTDIAG	HSDAG	FASTDIAG	HSDAG
1	1638	3354	1260	2996	1740	3023
2	2013	6646	1710	3167	2050	3203
3	2262	12106	1970	9454	2330	9544
4	2434	12355	2180	9536	2580	9654
5	2637	28111	2341	12044	2790	12165
10	3417	69950	2921	64631	3330	65240
20	4758	75317	3911	90715	5010	91726
all	46785	>100000	17301	>100000	10541	>100000

Table 4: Evaluation of FASTDIAG and HSDAG with the *Xerox* feature model from S.P.L.O.T.

Feature Model	#Variables	#Constraints	Redundancy Rate	Runtime (ms)
Car Selection	72	96	0.64	5070
SmartHome V. 2.2	61	63	0.29	1907
Xerox	172	205	0.71	3261

Table 5: Evaluation of FMCORE with selected S.P.L.O.T. feature models.

rate (see Formula 1) which was categorized in $\{2\%, 5\%, 7\%\}$. We used a random variable to control the degree of generated inconsistencies (the number of conflicts) in a feature model. As reasoning engine we used the CHOCO constraint solving library.[4] In order to import feature models to our environment we implemented a parser that generated CHOCO knowledge bases from S.P.L.O.T. SXFM based feature models.

$$Inconsistency\ Rate = \frac{\#conflicts\ in\ FM}{\#constraints\ in\ FM} \quad (1)$$

The performance tests were executed within a Java application running on a 64bit Windows 7 desktop PC using 8GB RAM and an Intel(R) Core(TM) i5-2320 CPU with 3.0GHz. Each run of the diagnosis algorithm for a specific setting has been repeated 10 times were in each run the ordering of the constraints was randomized. For each setting we evaluated the runtime (in ms) of both, the standard hitting set based approach to the termination of diagnoses [Reiter, 1987] (HSDAG) and FASTDIAG. As scenario we choose the diagnosis of *void feature models* where we induced different degrees of inconsistency (based on the *inconsistency rate measure* – see Formula 1). The upper bound for the evaluation time was set to 100.000 ms – in the case that this upper limit was exceeded, the search was stopped.

If one or a few diagnoses are required (which is typical for interactive settings) then FASTDIAG outperforms the standard HSDAG approach in most of the cases. If all diagnoses are required, for example, in situations where diagnoses are computed offline, the standard HSDAG approach seems to be the better choice. We want to emphasize that the presented diagnosis algorithms are independent of the underlying reasoning mechanisms, i.e., beside using a basic constraint-based approach for supporting the reasoning tasks (mainly consistency checking), description logics or SAT-based approaches can be applied as well. Finally, we also evaluated the performance of the redundancy detection algorithm FMCORE (see Table 5). Our goal was to figure out for the selected feature models to which extent the constraints in the feature models are redundant. We measured redundancy in the terms of the *redundancy rate* (see Formula 2).

$$Redundancy\ Rate = \frac{\#redundant\ constraints\ in\ FM}{\#constraints\ in\ FM} \quad (2)$$

The outcome of this analysis was that all the investigated feature models showed quite different degrees of redundancy (see Table 5). However, we consider these as preliminary results and further analyses have to be conducted, for example, we are interested in intra-constraint redundancies and the share of redundancy in cross-tree constraints with regard to the overall number of constraints in the feature model.

Note that the FMCORE algorithm is especially useful in situations where models are developed by one or a few engineers. In this case the degree of redundant constraints in the model is low. For scenarios with high redundancy rate, alternative algorithms have already been developed (see, e.g., [Felfernig *et al.*, 2011]).

6 Conclusions

In this paper we presented a consistency-based approach to explaining anomalies in feature models. We introduced definitions which are useful for the explanation of anomalies and discussed the corresponding algorithms which help to determine minimal diagnoses (FASTDIAG) and minimal sets of non-redundant constraints (FMCORE). Our future work will focus on: (1) The definition of further anomaly patterns in alternative knowledge representations such as *advanced feature models* [Batory, 2005] and UML models [Felfernig *et al.*, 2000]. Due to higher expressiveness, these representations include further anomaly patterns such as multiplicity

[4] www.emn.fr/z-info/choco-solver.

bounds which can not represented by configurations, unsatisfiable preconditions in constraints, and unexplained incompatibilities. (2) The development of mechanisms for the automated generation of test cases for feature models. (3) Further algorithms that enable the determination of diagnoses and redundancies on an intra-constraint level. (4) Evaluation of the developed algorithms with further benchmarks.

References

[Bakker *et al.*, 1993] R. Bakker, F. Dikker, F. Tempelman, and P. Wogmim. Diagnosing and solving over-determined constraint satisfaction problems. In *Proceedings of IJCAI-93*, pages 276–281. Morgan Kaufmann, 1993.

[Batory *et al.*, 2006] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Comm. of the ACM*, 49:45–47, 2006.

[Batory, 2005] D. Batory. Feature Models, Grammars, and Propositional Formulas. In H. Obbink and K. Pohl, editors, *Software Product Lines Conference*, volume 3714 of *LNCS*, pages 7–20. Springer, 2005.

[Benavides *et al.*, 2010] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35:615–636, 2010.

[Benavides *et al.*, 2013] D. Benavides, A. Felfernig, J. Galindo, and F. Reinfrank. Automated Analysis in Feature Modelling and Product Configuration. In *13th International Conference on Software Reuse (ICSR 2013)*, number 7925 in LNCS, pages 160–175, Pisa, Italy, 2013.

[Chandola *et al.*, 2009] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41:15:1–15:58, July 2009.

[Czarnecki *et al.*, 2005] K. Czarnecki, S.Helsen, and U.Eisenecker. Formalizing Cardinality-based Feature Models and their Specialization. *SoftwareProcess: Improvement and Practice*, 10(1):7–29, 2005.

[Felfernig *et al.*, 2000] 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*, 10(4):449–469, 2000.

[Felfernig *et al.*, 2004] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213 – 234, 2004.

[Felfernig *et al.*, 2011] A. Felfernig, C. Zehentner, and P. Blazek. Corediag: Eliminating redundancy in constraint sets. In *22nd International Workshop on Principles of Diagnosis*, pages 219–224, Murnau, Germany, 2011.

[Felfernig *et al.*, 2012] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *AI for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 26(1):53–62, 2012.

[Felfernig, 2007] A. Felfernig. Standardized configuration knowledge representations as technological foundation for mass customization. *IEEE Transactions on Engineering Management*, 54:41–56, 2007.

[Fleischanderl, 2002] G. Fleischanderl. Suggestions from the software engineering practice for applying consistency-based diagnosis to configuration knowledge bases. In *13th Intl. Workshop on Principles of Diagnosis (DX-02)*, pages 33–35, Semmering, Austria, 2002.

[Junker, 2004] U. Junker. QuickXPlain: preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI 2004, pages 167–172. AAAI, 2004.

[Kang *et al.*, 1990] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented Domain Analysis (FODA) – Feasibility Study. *TechnicalReport CMU – SEI-90-TR-21*, 1990.

[Mendonca and Cowan, 2010] M. Mendonca and D. Cowan. Decision-making coordination and efficient reasoning techniques for feature-based configuration. *Science of Computer Programming*, 75(5):311 – 332, 2010. Coordination Models, Languages and Applications(SAC 2008).

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Segura *et al.*, 2010] S. Segura, R. Hierons, D. Benavides, and A. Ruiz-Cortes. Automated test data generation on the analyses of feature models: A metamorphic testing approach. In *3rd Intl. Conference on Software Testing, Verification and Validation (ICST)*, pages 35–44, 2010.

[Trinidad *et al.*, 2008] P. Trinidad, D. Benavides, A. Duran, A. Ruiz-Cortez, and M. Toro. Automated error analysis for the agilization of feature modeling. *Journal of Systems and Software*, 81:883–896, 2008.

[Tsang, 1993] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.

[von der Massen and Lichter, 2004] T. von der Massen and H. Lichter. Deficiencies in Feature Models. In T. Mannisto and J. Bosch, editors, *Workshop on Software Variability Management for Product Derivation*, 2004.

[Wang *et al.*, 2010] B. Wang, Y. Xiong, Z. Hu, H. Zhao, W. Zhang, and H. Mei. A dynamic-priority based approach to fixing inconsistent feature models. In D. Petriu, N. Rouquette, and O. Haugen, editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *LNCS*, pages 181–195. Springer Berlin, 2010.

[White *et al.*, 2010] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094–1107, 2010.