

19th European Conference on Artificial Intelligence
ECAI 2010

Proceedings

Workshop on Configuration

Monday, Aug. 16, 2010

Lisbon, Portugal

Lothar Hotz

Alois Haselböck

WORKSHOP ORGANIZATION

Workshop Co-Chairs

Lothar Hotz, *HITeC e.V. University of Hamburg, Germany*
Alois Haselböck, *Siemens AG Österreich, Austria*

Program Committee

Patrick Albert, *IBM, France*
Claire Bagley, *Oracle Corporation, USA*
Alexander Felfernig, *Graz University of Technology, Austria*
Albert Haag, *SAP AG, Germany*
Alois Haselböck, *Siemens AG Österreich, Austria*
Lothar Hotz, *HITeC e.V. University of Hamburg, Germany*
Dietmar Jannach, *Technische Universität Dortmund, Germany*
Thorsten Krebs, *encoway, Germany*
Tomi Männistö, *Aalto University, Finland*
Klas Orsvarn, *Tacton System AB, Sweden*
Markus Stumptner, *University of South Australia, Australia*
Barry O'Sullivan, *Cork Constraint Computation Centre, Ireland*
Juha Tiihonen, *Aalto University, Finland*
Elise Vareilles, *Mines d'Albi-Carmaux, France*
Markus Zanker, *Universität Klagenfurt, Austria*

PREFACE

Configuration is the task of composing product models of complex systems from parameterisable components. This task demands for powerful knowledge-representation formalisms to capture the great variety and complexity of configurable product models. Furthermore, efficient reasoning 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 Configuration Workshop is to promote high-quality research in all technical areas related to configuration (e.g. constraint programming, description logics, rule-based systems, case-based reasoning, truth-maintenance systems, business processes, product line engineering, model-driven engineering).

The workshop is of interest for both researchers working in the various fields of applicable AI technologies and industry representatives interested in the relationship between configuration technology and the business problem behind product 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.

In this year, beside typical contributions about modeling and reasoning in configuration, two topics are focused: (1) The configuration beyond pure hardware applications, like configuration in software product lines, service configuration, model transformation, and configuration in operating systems. (2) The evaluation of configuration technologies in industrial settings. In particular, the latter is addressed by Michael Pirker (Siemens AG, Central Technology) in his invited talk “Industrial Configuration Problems: Possible Application Domains and Challenges”, with the abstract:

Industrial software-intensive systems often require some kind of configuration functionality. Market-ready, competitive products need to be equipped with a high degree of flexibility in terms of parameterisable software components. It can be observed that such components increasingly rely on well-formalized configuration models which help reducing the percentage of "hard-coded" software modules. Over the past 20 years substantial progress in model-formalization techniques and automated reasoning tasks, like for instance description logics models and reasoning, was made. Taking this into account, configuration-based solutions seem to be very interesting for a wide range of industrial domains.

In this talk, ideas for possible application examples, like for example in the domains of manufacturing automation, healthcare applications, or energy networks, will be presented together with the envisaged challenges to realize such configuration-enhanced solutions.

The 2010 workshop is the thirteenth in a series of successful Configuration workshops started at the AAAI'96 Fall Symposium and continued on IJCAI, AAAI, and ECAI since 1999. Beside researchers from a variety of different fields, past events always attracted a significant number of industrial participants from major configurator vendors and end-users. Its best papers, together with those of the 2009 workshop, will be collected in a special issue in AI EDAM in 2011.

Lothar Hotz and Alois Haselböck
August 2010

CONTENTS

Assessment of Benefits from Product Configuration Systems	9
<i>Lars Hvam, Anders Haug, and Niels Henrik Mortensen</i>	
Diagnosing Inconsistent Requirements	15
<i>Alexander Felfernig and Monika Schubert</i>	
Constraints filtering and evolutionary algorithm for interactive configuration and planning	21
<i>Paul Pitiot, Elise Vareilles, Michel Aldanondo, Mariem Djefel, and Paul Gaborit</i>	
A Flexible Approach to Product Configuration exploiting Fuzzy Knowledge and SQL	27
<i>Luigi Portinale and Massimiliano Bazzani</i>	
Constraint-based Modeling and Exploitation of a Vehicle Range at Renault's: Requirement analysis and complexity study	33
<i>Jean Marc Astesana, Yves Bossu, Laurent Cosserat, and H��l��ne Fargier</i>	
Modeling Technical Product Configuration Problems	40
<i>Andreas Falkner, Alois Haselb��ck, and Gottfried Schenner</i>	
A Generative Conguration Framework for Service Process Composition	46
<i>Rajesh Thiagarajan, Wolfgang Mayer, and Markus Stumptner</i>	
Encoding the Linux Kernel Configuration in Propositional Logic.....	51
<i>Christoph Zengler and Wolfgang K��chlin</i>	
Declarative Modeling at SAP Revisited – Lessons learnt when Applying Configuration Techniques.....	57
<i>Albert Haag</i>	
Software Product Lines Lessons learned when applying configuration techniques	60
<i>Marcos Didonet Del Fabro and Patrick Albert</i>	
Structured Development Process of Configuration Models	63
<i>Matthias Plietz</i>	

Assessment of Benefits from Product Configuration Systems

Lars Hvam, Anders Haug, Niels Henrik Mortensen¹

Abstract. This article presents an evaluation of the benefits obtained from applying product configuration systems based on a case study in four industry companies. The evaluation is based on theory and other case studies of the application of product configuration systems and how to assess the benefits to be obtained from applying product configuration systems. The impacts are described according to main objectives for implementing product configuration systems outlined in (Hvam et al, 2008):

- Lead time in the specification processes
- On-time delivery of the specifications
- Ressource consumption for making specifications
- Quality of specifications
- Optimization of products and services
- Other observations

The purpose of the study is partly to observe specific impacts observed from implementing product configuration systems in industry companies and partly to assess if the objectives suggested are appropriate for describing the impact of product configuration systems and identifying other possible objectives. The above-mentioned objectives focus on changes in the performance of the company's specification processes. The empirical study of the companies also gives an indication of more overall performance indicators affected by the use of product configuration systems e.g. increased sales, decrease in the number of SKU's, ability to introduce new products, cost reductions in production, purchasing, transport, installation and after sales service induced by improved performance in the specification processes and improved quality of the specifications.

Significance. Product configuration systems are increasingly used as a means for efficient design of customer tailored products, and this has led to significant benefits for industry companies. However, the specific benefits gained from product configuration are difficult to measure. This article discusses how to assess the benefits from the use of product configuration based on a suggested set of measurements and an empirical study of four industry companies.

Keywords. Mass Customization, product configuration, complexity management, key performance indicators

1 INTRODUCTION

Customers worldwide require personalised products. One way of obtaining this is to customise the products by use of product configuration systems (Tseng and Piller, 2003; Hvam et al 2008). Several companies have acknowledged the opportunity to apply IT-based configuration systems to support the activities of the product configuration process (see for example www.configurator-database.com/). Companies like Dell Computer and American Power Conversion (APC) heavily rely on the performance of their configuration systems, as a configuration of their complex product portfolio would not be feasible if the product configuration processes should be carried out manually (Tiihonen et al., 1996).

A product configuration system is capable of supporting the activities of specifying products in sales, design and methods engineering – the specification process. The activities in the specification processes include an analysis of the customer's needs, design and specification of a product variant which full-fill the customer's needs and specification of e.g. the product's manufacturing, transportation, erection on site and service (specification of the product's life cycle properties). The activities in the specification processes are characterized by having a relatively well-defined space of (maybe complex) solutions as a contrast to product development, which is a more creative process (Hvam et al, 2008, Hvam & Have, 1998).

Typical goals for the specification processes are the ability to find an optimal solution according to the customer's needs, high quality of the specifications, short lead time and a high productivity in the specification process.

This article focuses on the possibilities offered by product configuration systems as to the support of the specification processes and on how to assess the benefits to obtain from applying product configuration in the specification processes. In Section 3 suggested groups of targets for applying product configuration are listed. The targets focus specifically on benefits in the specification processes. Based on the empirical study and the referred theory, more overall targets covering other functions of the company or the total company will be suggested.

¹ Centre for Product Modelling, Technical University of Denmark.
www.man.dtu.dk, www.productmodels.org, www.sam.sdu.dk

2 LITERATURE STUDY

Most configuration literature focuses on technical solutions, methods and techniques, while only a minor part of this literature focuses on empirical studies of the benefits from applying product configuration systems. In the following, some of the literature with an empirical perspective is presented.

Barker and O'Conner (1989) describe the case of Digital Equipment Corporation (DEC). DEC uses configurators for validation of the technical correctness of customer orders and for guiding the actual assembly of these orders. They state "overall the net return to Digital is estimated to be in excess of \$40 million per year", and that the configurators are "contributing to customer satisfaction, lower costs, and higher productivity"; "insures that complete, consistently configured systems are shipped to the customer"; "simplifies field and manufacturing training needs and avoids confusion about new products which can delay time-to-market significantly"; "increases manufacturing's flexibility"; "increased the technical accuracy of orders entering manufacturing"; "assures that when the components of the order come together for the first time at the customer site the system will work"; and "major positive impact on cycle times, inventory levels, and manufacturing costs".

Ariano and Dagnino (1996) describe the case of a manufacturer of modular wooden office furniture who applies a configurator for the creation of bills of materials. They claim that the benefits achieved from the configurator are many: "a new and more organized way of structuring the company's product line"; "allows for a more consistent, faster, easier, and more comprehensive way to enter an order"; "while the order is entered, the system verifies that the configuration of the products is correct and compatible with the company's offerings"; "helps in quoting an accurate pricing to the company's products"; and "implies a reduction in the duplication of information, pricing deviations, and configuration inconsistencies".

Fleischanderl et al. (1998) describe the use of a configurator for configuring large telecommunication systems. They claim that the configurator: has "improved the quality of the configuration results"; helps "avoiding error-prone manual editing of parameters"; has "revealed numerous errors, such as cables having wrong length codes"; and "makes the knowledge about the EWSD [telecommunication systems] configuration explicit".

Forza and Salvador (2002a) describe the case of a small company producing voltage transformers. They mention the effects of the use of a configurator: a "reduction to almost zero of the errors in the configurations released by the sales office"; "reducing the total time necessary for generating the tender"; made it "possible to recover a notable volume of man-hours, which freed part of the sales personnel for tasks with greater additional value"; "made it possible to increase technical productivity, both as regards product documentation release and design activities"; an "increase in technical department productivity"; a "formalisation of the company knowledge"; and enabling "the transformation of individual competencies into organisational competencies".

Forza and Salvador (2002b) describe a project of implementing product configuration software into a small manufacturing company producing mould-bases for plastics moulding and punching-bases for metal sheet punching. They claim that two main kinds of advantages have been achieved: (1) "reduction of manned activities in the tendering process (tendering lead-time from 5–6 to 1 day)"; and (2) "increase in the level of correctness of product information (almost 100%)". They argue that the case study shows that the company obtained: a rapid payback of the investment in configuration technology; a competitive advantage; and better inter-firm co-ordination.

Based on studies of twelve Danish firms that were using product configurators, Pedersen and Edwards (2004) present the results of the twelve companies' answers to the realized effects of their configurator projects. In the study, the firms were to estimate effects by giving scores from 1 to 5, where 1 equals "very small", and 5 equals "very large", while 0 equals "without influence". The three top scorers are: improved quality (avg. ~ 4.4); lower turnaround time (avg. ~ 3.6); and less use of resources (avg. ~ 3.3).

Forza et al. (2006) describe the case of a company that produces electric motors. They state that the configurator: "enhances product assortment communication"; "makes it easier and faster to explore the solution space offered by the company"; "enables a faster, accurate generation of a feasible offer without consulting the technical office"; "enables a faster, accurate creation of product code, BOM, and production cycle"; "allows storage of a large amount of customer data collected during the exploration and configuration phases"; and "allows rapid retrieval of past configurations for maintenance or repair purposes".

Petersen et al. (2007) describe the case of Aalborg Industries, who makes steam and heat generating equipment for maritime and industrial applications. Petersen et al. (2007) state that because of the (sales) configurator the company is: "gaining significant benefits, and has learned much about the challenges of implementing product configuration in ETO".

Hong et al. (2008) describe the case of Gienow Windows and Doors, where a configurator is used for: modelling the designs based on customer needs; creating requirements of materials, machines, and personnel; and identifying the optimal production schedule. They claim that "the lead time from a customer order to the product delivery has been reduced to 3 weeks compared to the average of 2 months in this industry".

Ladeby (2009) describes the configurator project at NNE Pharmaplan, who uses a configurator for 3D visualisation system of plant layouts. It is stated that a main benefit of the system is that "a customer does not have to wait for weeks before he sees drawings and illustrations of what has been agreed upon".

Ladeby (2009) describes the configurator project of GEA Niro, who designs and supplies spray drying plants. The configurator of GEA Niro focuses on the quotation phase, and it is used in about 50 percent of the first quotations sent out to customers. He states that: "the process of making quotations has become more standardised end formalised"; "product knowledge has become more standardised"; the sales person "gets the whole quotation served on a plate and sends it to the

customer"; and "preservation of knowledge has been a motivation for the configurator project".

3 HYPOTHESIS

The literature refers to numerous examples and a few surveys on the impact of applying product configuration systems. However, the benefits are described in many different ways, and it is often uncertain whether the benefits claimed have been obtained from the product configuration system or from other initiatives in the company. In order to make a more specific assessment of benefits obtained a list of suggested benefits from applying product configuration in the specification processes has been made. The suggested measurements include:

- Lead time in the specification processes
- On-time delivery of the specifications
- Ressource consumption for making specifications
- Quality of specifications
- Optimization of products and services in the specification processes

Lead time refers to the interval of time from when a specification process is initiated until a finished specification is available. An example is the number of days from when a customer makes an enquiry until the customer receives an offer.

On-time delivery for specifications is defined as the number of specifications out of the total number of specifications which are completed within the agreed time span. On-time delivery is normally specified as the percentage of the total specifications which are completed at the agreed time. An example is when working out offers, where the company has promised the customers that they can always expect to receive an offer within at most 3 working days. A sample of 100 random offers shows that 45 of them are delivered within 3 working days, while 55 are delivered after 4 working days or later. In this case, the percentage of on-time delivery for offers is 45%.

Resource consumption for making specifications. The frequency of the individual specification activities, combined with the duration (use of man-hours) of the individual specification activities, reflects where the largest use of resources in the specification task lies, and where uniform tasks are executed with high frequency.

In order to be able to reveal the use of resources and find uniform tasks that are performed with high frequency, an analysis can be made of the activities performed in the specification process. The analysis can be carried out as a frequency study to find out how much of employees' time is spent on given tasks – defined in terms of the specification result or the specification method (activity).

Quality of specifications can be defined in several ways. One aspect is understandability/readability of the specifications, for example whether or not a customer understands the central elements in an offer he has received, or whether or not the production engineer understands the design drawings on which he is to base production. The basic question here is if the specification in question is able to pass on to the receiver an unambiguous and complete description, for example of the product's design. This aspect of a specification's quality is obviously difficult to measure, both because it can be a question of subjective evaluation on the part of the receiver, and because receivers of the specification can have different backgrounds for interpreting a specification.

Another aspect of quality is the number of errors. Errors in specifications can be defined as the proportion of the specifications containing errors. Here, errors are defined as those errors that, if they are not discovered, will lead for example to manufacture of a faulty product – so such errors as insignificant typos are not to be counted. An example is the number of lists of parts with errors, compared to the total number of lists of parts produced. Another example is the number of offers in which the pre-calculated cost price differs by more than 5% from the cost price arrived at by post-calculation.

Optimization of products and services in the specification processes. Using a configuration system makes it possible to optimize products in relation to the customer's requirements, or for example in relation to production costs or maintenance/service.

The suggested targets and how to exactly measure each of the targets are further elaborated in (Hvam et al 2008). The impact from product configuration has been studied in four different companies based on the suggested measurements.

Besides these targets for the specifications processes other targets measuring the more overall impact from product configuration would be relevant to include. Based on the literature and observations made from the case study, possible candidates for measuring the overall impact of product configuration systems will be identified and discussed.

4 CASE STUDY

4.1. Impacts from product configuration

In the following we shall give a brief introduction to the 4 industry companies of this case study, of their configuration projects and discuss the impacts observed.

4.1.1. Company A

Company A is an engineering and industrial company with an international market leading position within the area of development and manufacturing of cement plants. The company has a turnover around 1 billion USD.

A modern cement plant typically produces 2-10,000 tonnes of clinkers per day (TPD), and the sales price for a 4,000 TPD plant is approx. 100 million USD². Every complete cement plant is customized to suit the local raw material and climatic conditions, and the lead-time from signing the contract to start-up is around 2½ years.

The company has implemented and used a configuration system since 2000 to support the quotation process. The first version of the configuration system was implemented on a budget at approx. 800,000 Euros. Today the company has a configuration team with 4-5 employees responsible for implementation and running of 10-12 configuration systems used in sales and engineering. The quotation process is carried out in two steps. The first step is a so-called budget quotation, including an overall dimensioning of the cement factory, a process diagram and a price estimate. The next step is a so-

² A 4,000 tonnes per day (TPD), complete kiln line, semi turn-key, service, supervision, vehicles, training, steel plates to local manufacturer, and civil design.

called detailed quotation, including a detailed description of the processes and machines in the cement factory. The configuration project focused on the budget quotation because the budget quote included fewer details, and because all significant decisions as to the cement factory's capacity, emissions, total project costs etc. are made during the budget quotation. The process analysis revealed that the process of making budget quotations was very resource consuming, with a long lead time and leading to quotations of varying quality. A gap analysis indicated that the lead time for making budget quotations could be reduced from 3-5 weeks to 1-2 days, the resources spent could be reduced from 15-25 man-days to 1-2 man-days and, finally, by using a product configuration system it would be possible to optimize the cement factory with respect to e.g. capacity, emissions, price and the use of previously designed machines. The impacts observed are outlined in the table below:

Table 1. Observed impacts from product configuration in company A

Lead time	Lead time for making quotations reduced from 3-5 weeks to 1-2 days.
On-time delivery	Now 95-100 % of quotations are delivered on time. Before, only 50 % of the requests were even responded to by a quotation, now 100 % gets a quotation.
Resource consumption	Resources for making quotations reduced with 50 %.
Quality of specifications	The quotations become more uniform and of better quality. More accurate calculation of sales prices.
Optimization of products/ and services	Possible to simulate different solutions to the customer. More structured negotiations with the customers. Possible to optimize the plant with respect to increased use of previously engineered and produced equipment.
Other observations	The configuration system ensures that the sales man obtains all necessary information before the quotation is made. This leads to an improved quality of the quotations and the subsequent engineering process. Application of the product configuration system has led to an increase in the sales of more standardized machines, which leads to significant savings in the engineering, production and erection on site

The suggested metrics give an unambiguous measurement of the benefits realized in the specification processes. Lead time, on-time delivery and resource consumption were possible to measure exactly. The quality issue was assessed by comparing the content of the quotations generated from the configuration systems with quotations made outside the configuration system.

The suggested measurements focus on the quotation process. However, the company claims that other benefits, which are more significant than the improvements in the quotation process, have been obtained. These benefits include an increase in sales as all requests are now being responded to with

a quotation and a reduction of costs in engineering, production and erection on site. Even though these benefits have been measured, it is not possible to state how much the configuration system in itself has contributed to these benefits and how much come from other initiatives, such as product redesign, reengineering of business processes or improvement/ implementation of other IT-systems.

4.1.2. Company B

Company B is an international engineering company with a market leading position within the area of design and supply of spray drying plants. The company is creating approx. 340 mio. USD in turnover a year. The products are characterized as highly individualized for each project.

The configuration system was implemented in company B in 2004 and it is in many ways similar to the configuration system of company A. Today, the company has a configuration team with 10-12 employees running the configuration system and doing external configuration projects for other companies in the industry group. The project focuses on the quotation process. The aim of introducing a product configuration system is to reduce lead times and resources spent on making quotations, optimization of the spray drying plants and the formalisation of product knowledge, in order to make it accessible to relevant persons in the organisation. The impacts observed are outlined in table 2 below.

Table 2. Observed impacts from product configuration in company B

Lead time	Lead time for making quotations reduced from 3-5 days to 2 hours.
On-time delivery	Between 95 and 100 % after implementing the configuration system.
Resource consumption	Resources used for quotations reduced from 20 to 2 hours per quotation
Quality of specifications	The quotations become more uniform and of better quality. More accurate calculation of sales and cost prices.
Optimization of products/ and services	Mass flow diagrams and process simulation is integrated with the configuration system, which makes it possible to optimize the performance of the plant and single machines.
Other observations	The modelling of the products for the configuration system has led to an increased formalization of engineering knowledge. Increased sales due to a more efficient quotation process. Shorter total lead time. More standardized products in the projects leading to reduction of project costs.

Application of the product configuration system has led to a reduction of lead times and resources spent in the quotation process. On-time delivery and quality of the quotation process have been improved as well. It has been possible to measure those benefits, and it is clear that these benefits come from the application of the product configuration. As with company A, other and more overall benefits have been observed like reduction of costs in engineering, production and

installation due to sales of more standardized products have also been observed. However, it is not possible to define how much the configuration system has contributed to these benefits.

4.1.3. Company C

Company C produces data centre infrastructure such as uninterruptible power supplies, battery racks, power distribution units, racks, cooling equipment, accessories etc. The total turnover is approx. 4 billion USD (2008). Company C has implemented and used product configuration systems since 2000. Today, Company C has 8-9 product configuration systems. The company has formed a configuration team with approx. 25 employees. The configuration team is responsible for development and maintenance of the product configuration systems, which are used worldwide.

The product configuration systems are an integrated part of the company's business setup. The products are sold through the product configuration systems, which makes it possible for the company to control a huge amount of sales personnel and agents around the world. The product configuration, which includes working out quotations and manufacturing specifications, is carried out by the configuration system, thereby saving considerably resources. The lead time for making quotations and manufacturing specifications is reduced significantly. And finally, the product configuration systems make it easier to introduce new versions of the products to the sales personnel and the customers.

Table 3. Observed impacts from product configuration in company C

Lead time	Lead time for making quotations and Bills of Material (BOM's) and routes reduced from 3-5 days to less than 1 hour.
On-time delivery	100 % for quotations and BOM's and routes generated by the configuration system.
Resource consumption	Resources used for making quotations and manufacturing (BOM's and routes) reduced to less than 10 % of previously used.
Quality of specifications	Specifications coming from the configuration system have significant less errors than "hand-made" specifications, leading to less costs in production and installation.
Optimization of products/ and services	The use of the configuration system has made it possible to keep down the number of items maintained the company's ERP-system. The company has estimated that the cost of an item number is approx. 10,000 USD in its life time
Other observations	The company is heavily focusing on keeping down complexity costs and sees the use of product configuration systems as part of an overall business strategy also including market focus and product modularization. Total lead time from sales to delivery and installation reduced from 400 days to 16 days. Increased ability to introduce new products via the configuration systems.

Company C has realized significant improvements of the sales and ordering processes from applying the product configuration system. As with Company A and B, it has been possible to measure or at least assess those benefits, and it is clear that those benefits come from the product configuration system, as the configuration systems now generate the specifications. In company C, the use of product configuration systems is a part of an overall business model also including a modularized product assortment, a focused market strategy and a supply chain based on mass production of standard modules and assembly of customer tailored products. The total business set-up has led to significant improvements of productivity, quality and delivery times. In this context the configuration system is a necessary part in order to achieve the total benefits.

4.1.4. Company D

Company D is making electronic switchboards. It has more than 100 employees and a turnover of approx. 15 million Euros. The process analysis revealed that the lead-time for generating quotations was 3 to 5 days, and the company uses 2 to 4 man-hours for each quotation. The process leads to frequent errors, and often the time necessary for the optimization of the boards cannot be found.

Table 4. Observed impacts from product configuration in company D

Lead time	Lead time for making quotations and BOM's reduced from 3-5 days to 10 minutes.
On-time delivery	On-time delivery for quotations and BOM's generated from the configuration system 100 %.
Resource consumption	Resources used for making quotations and manufacturing BOM's reduced from 2-4 hours to 10 minutes. The resources are reduced to zero for the customers using the configuration system directly.
Quality of specifications	Significant reduction of errors in quotations and BOM's by using the configuration system.
Optimization of products/ and services	The configuration system makes it possible to optimize the products with respect to heat loss, component fabricates and vacant space.
Other observations	The modelling of the switch boards for the configuration system gave rise to an evaluation of the components used when designing the switch boards to the customers. This has led to an optimization of components used with respect to number of items, costs and performance. More error-free specifications in manufacturing led to improved productivity.

By implementing a product configuration system the company gets a much more structured process flow, where the company's knowledge regarding construction of an electronic switch board is made available to the customers, and complex

calculations can be made very quickly. The desired effects for the company of the application of a product configuration system are identified as:

- A significant reduction of lead time for making quotations from 3-5 days to 10 minutes.
- A total elimination of resources spent for making quotations, as the customers can now configure an electronic switchboard on their own by using the product configuration software available on the company's homepage.
- The opportunity of optimizing the electronic switchboards with respect to e.g. heat loss and price.

In company D the effects on the sales and ordering processes of applying product configuration could be measured unambiguously, except from the quality and optimization, where the improvements were assessed based on a study of the specifications generated before and after using the configuration system. Being a small company, those effects have a significant impact on the company's total performance. Modelling the products for the configuration system gave rise to improvements of the products. More error-free specifications have contributed to improved productivity in manufacturing.

5 CONCLUSION

The study of the four cases has shown that the application of product configuration systems may lead to significant benefits. The suggested measurements focusing on the specification processes have been tested. The case study shows that it is possible to measure those indicators and also that those improvements can be linked to the application of product configuration systems. However, further work is needed in order to clarify in more detail how quality and optimization of products and services can be measured.

Besides the specific measurements on the specification processes, the study of the four companies has made it clear that other and even more significant benefits have been achieved from applying product configuration systems. These benefits include:

- Increased sales
- Reduction of costs in e.g. production due to sales of more standardized products and more error free specifications
- Reduction of e.g. engineering costs caused by a collection of all needed information via the configuration system
- Formalization of engineering knowledge
- Reduction of item numbers

Further work is needed in order to clarify how these benefits can be measured in more detail, and how the relative contribution from the configuration system to those overall benefits may be documented. This also includes an investigation of how the product configuration system is seen as a part of an overall business strategy, thus being a needed brick in the puzzle in order for the company to realise its business strategy.

6 REFERENCES

- Ariano, M. & Agnino, D. (1996). An intelligent order entry and dynamic bill of materials system for manufacturing customized furniture. *Computers and Electrical Engineering*, 22(1), 45-60.
- Barker, V.E., O'Connor, D.E., Bachant, J. & Soloway, E. (1989). Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM*, 32(3), 298-318.
- Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H. & Stumptner, M. (1998). Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems*, 13(4), 59-68.
- Forza, C. & Salvador, F. (2002a). Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems. *International Journal of Production Economics*, 76(1), 87-98.
- Forza, C. & Salvador, F. (2002b). Product Configuration and inter-firm co-ordination: An innovative solution from a small manufacturing enterprise. *Computers in Industry*, 49(1), 37-46.
- Forza, C., Trentin, A. & Salvador, F. (2006). Supporting product configuration and form postponement by grouping components into kits: The case of MarelliMotori. *International Journal of Mass Customization*, 1(4), 427-444.
- Gilmore, J. & Pine J. (2000); *Markets of One, - Creating Customer Unique Value through Mass Customization*, Harvard Business Review, Boston, 2000.
- Hammer, M. (1990): Re-engineering work: don't automate, obliterate, Harvard Business Review, July-August, 1990.
- Hvam L., Have U.; Re-engineering the Specification Process; *Business Process Management Journal*, Vol. 4, No 1 1998, p. 25- 43. ISSN 1355 2503.
- Hvam L., Mortensen N.H., Riis J., *Product Customization*; Springer, ISBN-10: 3540714480 ISBN-13: 978-3540714484. January 2008.
- Hong, G., Hu, L., Xue, D., Tu, Y.L. & Xiong, Y.L. (2008). Identification of the optimal product configuration and parameters based on individual customer requirements on performance and costs in one-of-a-kind production. *International Journal of Production Research*, 46(2), 3297-3326.
- Ladeby, K.R. (2009). *Applying Product Configuration Systems in Engineering Companies: Motivations and Barriers for Configuration Projects*. PhD Thesis. Lyngby, Denmark: Technical University of Denmark, Department of Management Engineering and Operations Management.
- Pedersen, J.L. & Edwards, K. (2004). Product configuration systems and productivity. *Proceedings of International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems (PETO)*, pp. 165-176.
- Petersen, T.D., Jorgensen, K.A., Hvolby, H.H. & Nielsen, J.A. (2007). Multi level configuration of ETO products. *4th International Conference on Product Lifecycle Management: Assessing the Industrial Relevance*, pp. 293-302.
- Tiihonen, J., Soininen, T., Männistö, T. & Sulonen, R. (1996). State-of-the-practice in Product Configuration – a Survey of 10 Cases in the Finnish Industry. In *Knowledge Intensive CAD*, volume 1 (Ed.: Tomiyama, T., Mäntylä, M. & Finger S.). Chapman & Hall, pp. 95-114.
- Tseng Mitchell M. and Piller, Frank. T. eds(2003); *The Customer Centric Enterprise – Advances in Mass Customization and Personalization*; Springer Verlag. ISBN 3-540-02492-1.

Diagnosing Inconsistent Requirements

Alexander Felfernig¹ and Monika Schubert¹

Abstract. Knowledge-based configurators are supporting configuration tasks for complex products such as telecommunication systems, computers, or financial services. Product configurations have to fulfill the requirements articulated by the user and the constraints contained in the configuration knowledge base. If the user requirements are inconsistent with the constraints in the configuration knowledge base, users have to be supported in finding out a way from the *no solution could be found* dilemma. In this paper we introduce a new algorithm that allows the determination of personalized diagnoses for inconsistent user requirements in knowledge-based configuration scenarios. We present the results of an empirical study that show the advantages of our approach in terms of prediction quality.

1 Introduction

On an informal level, *configuration* can be defined as a "*special case of design activity, where the artifact being configured is assembled from instances of a fixed set of well-defined component types which can be composed conforming to a set of constraints*" [15]. Configuration systems typically exploit two different types of knowledge sources: on the one hand the explicit knowledge about the *user requirements*, on the other hand deep *configuration knowledge* about the underlying product. Configuration knowledge is represented in the form of a product structure and different types of constraints [5] such as *compatibility constraints* (which component types can or cannot be combined with each other), *requirements constraints* (how user requirements are related to the underlying product properties), or *resource constraints* (how many and which components have to be provided such that needed and provided resources are balanced).

Interacting with a knowledge-based configurator typically means to *specify* a set of requirements, to *adapt* inconsistent requirements, and to *evaluate* alternative configurations (solutions). In this paper we focus on a situation where the configurator is *not able to find a solution*. In such a situation it is very difficult for users to find a set of changes to the specified set of requirements such that a configuration can be found [6]. In order to better support users, we introduce PERSDIAG which is an algorithm for the *personalized diagnosis of inconsistent user requirements*. PERSDIAG improves the precision of diagnosis predictions (see Section 5).

State-of-the-art approaches to the determination of minimal diagnoses for inconsistent user requirements are focusing on minimal-cardinality diagnoses [6] or on the pre-calculation of all possible diagnoses [12]. In the context of recommender systems [7], the complement of such a diagnosis is often denoted

as *maximally successful sub-query* [9, 12]. Such a query consists of those elements which are not part of a corresponding minimal diagnosis. In the context of constraint-based systems [18] diagnoses are also interpreted as *explanations* [13].

Especially in interactive settings the calculation of all possible diagnoses is infeasible due to unacceptable runtimes [8]. Furthermore, it cannot be guaranteed that minimal-cardinality diagnoses lead the most interesting explanations for a user [8, 13]. The work of [13] is a first step towards the tailoring of the presented set of diagnoses in the sense that so-called *representative explanations* are determined. These explanations fulfill the criteria that each element part of a diagnosis is also contained in at least one of the diagnoses presented to the user. The work presented in [8] takes one further step towards this direction by introducing concepts that allow to determine personalized repair actions for inconsistent requirements in knowledge-based recommendation [1] where – in contrast to knowledge-based configuration scenarios – there exists a fixed and predefined set of candidate products.

On the basis of related work in the field, we introduce a new algorithm for the personalized diagnosis of inconsistent user requirements which is especially tailored to knowledge-based configuration scenarios. The algorithm (PERSDIAG) performs a best-first search for diagnoses acceptable for the user where the decision on which nodes to expand during search is based on criteria often used in recommender systems development [7]. The major contribution of this paper is to show how standard model-based diagnosis approaches [2, 14] can be extended with intelligent personalization concepts that improve the *prediction quality of diagnosis selection*.

The remainder of this paper is organized as follows. In Section 2 we introduce a working example which will be used for illustration purposes throughout the paper. In Section 3 we discuss a basic approach to identify inconsistent user requirements [6] which is based on the concepts of model-based diagnosis [2, 14]. In Section 4 we present an algorithm (PERSDIAG) for the personalized identification of minimal sets of inconsistent user requirements. The results of an empirical evaluation are presented in Section 5. In Section 6, we discuss related work. The paper is concluded with Section 7.

2 Working Example

We will use *computer configuration* as a working example throughout this paper. The task of identifying a configuration for a given set of user requirements can be defined as follows (see Definition 1). This definition is based on [6] and – in contrast to the component-port based representation of a configuration problem [6] – it relies on the definition of a Constraint Satisfaction Problem (CSP) [18].

Definition 1 (Configuration Task): A configuration task can be defined as a CSP (V, D, C) , where $V =$

¹ Graz University of Technology, Institute for Software Technology (IST), Applied Software Engineering (ASE), Austria, email: {felfernig, schubert}@ist.tugraz.at - an extended version of this paper has been submitted to the AIEDAM Special Issue on Configuration 2010.

$\{v_1, v_2, \dots, v_n\}$ is a set of finite domain variables and $D = \{dom(v_1), dom(v_2), \dots, dom(v_n)\}$ represents the domain of each variable v_i . $C = C_{KB} \cup C_R$ is a set of all constraints, which can be divided into the configuration knowledge base $C_{KB} = \{c_1, c_2, \dots, c_m\}$ and the set of specific user requirements $C_R = \{c_{m+1}, c_{m+2}, \dots, c_p\}$.

A simple example for a configuration task (V, D, C) is $V = \{cpu, graphic, ram, motherboard, harddisk, price\}$ where *cpu* is the type of central processing unit, *graphic* represents the graphics card, *ram* represents the main memory specified in GB, *motherboard* represents the type of motherboard, *harddisk* is the harddisk capacity in GB, and *price* represents the overall price of the computer. These variables fully describe the potential set of requirements that can be specified by the user. The respective variable domains are $D = \{dom(cpu) = \{CPUA, CPUB\}, dom(graphic) = \{GCA, GCB, GCC, GCD\}, dom(ram) = \{1, 2, 3, 4\}, dom(motherboard) = \{MBX, MBY, MBZ, MBW\}, dom(harddisk) = \{200..700\}, dom(price) = \{300..600\}\}$. Note that for reasons of simplicity we do not explicitly discuss pricing constraints – the reader can assume that for each relevant variable value there is a corresponding specified price and that there is a set of constraints responsible for calculating the overall price of the configuration. The set of possible combinations of variable instantiations is restricted by the constraints in the configuration knowledge base $C_{KB} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$. In our working example these are simplified technical & sales constraints:

- $c_1: cpu = CPUA \Rightarrow graphic \neq GCA$
- $c_2: cpu = CPUB \Rightarrow ram > 1$
- $c_3: motherboard = MBY \Rightarrow ram > 1$
- $c_4: harddisk = 700 \Rightarrow motherboard = MBW$
- $c_5: motherboard = MBX \Rightarrow graphic = GCB \vee graphic = GCD$
- $c_6: motherboard = MBX \Rightarrow ram = 1 \vee cpu \neq CPUA$

For the purposes of our simple example, we assume that the following requirements have been specified by the user ($C_R = \{c_7, c_8, c_9, c_{10}, c_{11}, c_{12}\}$):

- $c_7: cpu = CPUA$
- $c_8: graphic = GCA$
- $c_9: ram \geq 2$
- $c_{10}: motherboard = MBX$
- $c_{11}: price \leq 350$
- $c_{12}: harddisk \geq 200$

Based on this example of a configuration task we can introduce a definition of a concrete configuration, i.e., a solution for a configuration task.

Definition 2 (Configuration): A configuration for a given configuration task (V, D, C) is an instantiation $I = \{v_1 = i_1, v_2 = i_2, \dots, v_n = i_n\}$ of each variable v_j where $i_j \in dom(v_j)$. A configuration is *consistent* if the assignments in I are consistent with the constraints in C . Furthermore, a configuration is *complete* if all the variables in V are instantiated. Finally, a configuration is *valid*, if it is both consistent and complete.

In our working example, we assume that users already interacted with the computer configurator and created several configurations ($CONFIGS = \{conf_1, conf_2, conf_3, conf_4, conf_5, conf_6, conf_7\}$). These configurations are stored in a corresponding table (see Table

1). We will exploit this information for the determination of personalized diagnoses in Section 4.

Table 1. User interaction data from configuration sessions.

	CPU	GRAPHIC	RAM	MB	HD	PRICE
<i>conf₁</i>	CPUA	GCB	1	MBX	200	350
<i>conf₂</i>	CPUB	GCA	3	MBY	500	400
<i>conf₃</i>	CPUA	GCD	1	MBX	200	450
<i>conf₄</i>	CPUA	GCC	3	MBZ	650	550
<i>conf₅</i>	CPUB	GCB	3	MBW	700	600
<i>conf₆</i>	CPUA	GCC	2	MBY	200	300
<i>conf₇</i>	CPUB	GCC	4	MBY	300	550

3 Calculating Minimal Cardinality Diagnoses

For the example configuration task specified in Section 2 we are *not* able to find a valid solution, for example, the processor type *CPUA* is incompatible with the graphic card *GCA* (a simple sales constraint). Therefore we want to identify the minimal set of requirements ($c_i \in C_R$) which have to be relaxed or adapted in order to find a solution. For identifying such minimal sets, we exploit the concepts of Model-Based Diagnosis (MBD) [2, 14]. MBD starts with a system description which in our case encompasses the configuration knowledge base C_{KB} that describes the set of possible product configurations. If the actual behavior of the system conflicts with its intended behavior (a corresponding configuration can be identified), the task of a diagnosis component is to determine those elements (in our case the elements are requirements in C_R) which, when assumed to be functioning abnormally, sufficiently explain the discrepancy between the actual and the intended behavior of the system. A diagnosis is a minimal set of faulty components (requirements) that need to be relaxed or adapted in order to be able to identify a configuration.

On a more technical level, minimal diagnoses for faulty user requirements can be identified as follows. Let us assume the existence of a set $C_{KB} = \{c_1, c_2, \dots, c_m\}$ of configuration constraints and a set $C_R = \{c_{m+1}, c_{m+2}, \dots, c_p\}$ of user requirements (represented as constraints) *inconsistent* with C_{KB} , i.e., no solution can be found for the constraints in $C_R \cup C_{KB}$. In such a situation, state-of-the-art configurators [17] calculate a set of minimal diagnoses $DIAGS = \{diag_1, diag_2, \dots, diag_k\}$, where $\forall diag_i \in DIAGS: C_{KB} \cup (C_R - diag_i)$ is consistent. A corresponding *User Requirements Diagnosis Problem (UR Diagnosis Problem)* can be defined as follows:

Definition 3 (UR Diagnosis Problem): A User Requirements Diagnosis (UR Diagnosis) Problem is defined as a tuple (C_{KB}, C_R) where C_{KB} represents the configuration knowledge base and C_R is a set of user requirements.

Based on the definition of the UR Diagnosis Problem, a *UR Diagnosis* can be defined as follows:

Definition 4 (UR Diagnosis): A User Requirements Diagnosis (UR Diagnosis) for (C_{KB}, C_R) is a set of constraints $diag \subseteq C_R$ such that $C_{KB} \cup (C_R - diag)$ is consistent. A diagnosis $diag$ is minimal if there does not exist a diagnosis $diag' \subset diag$ s.t. $C_{KB} \cup (C_R - diag')$ is consistent.

Following the basic principles of Model-Based Diagnosis (MBD) [2, 14], the calculation of diagnoses is based on the

identification and resolution of conflict sets. A *conflict set* in the user requirements C_R can be defined as follows:

Definition 5 (Conflict Set): A Conflict Set is defined as a subset $CS \subseteq C_R$ s.t. $CS \cup C_{KB}$ is inconsistent. CS is minimal iff not exists a conflict set CS' with $CS' \subset CS$.

In our simple working example, the user requirements $C_R = \{c_7, \dots, c_{12}\}$ are inconsistent with the constraints in the configuration knowledge base $C_{KB} = \{c_1, \dots, c_6\}$, i.e., there does not exist a configuration (solution) that completely fulfills the requirements in C_R . The minimal conflict sets are $CS_1 = \{c_7, c_8\}$, $CS_2 = \{c_8, c_{10}\}$, and $CS_3 = \{c_7, c_9, c_{10}\}$ since each of these conflict sets is inconsistent with the configuration knowledge base and there do not exist conflict sets CS_1' , CS_2' , and CS_3' with $CS_1' \subset CS_1$, $CS_2' \subset CS_2$, and $CS_3' \subset CS_3$.

In model-based diagnosis (MBD) [2, 14] the standard algorithm for determining minimal diagnoses is the *hitting set directed acyclic graph* (HSDAG) as described in [14]. User requirements diagnoses $diag_i \in DIAGS$ can be calculated by resolving conflicts in the set of requirements C_R . Due to its minimality property, one conflict can be resolved by deleting exactly one of the elements from the conflict set. After deleting at least one element from each identified conflict set we are able to present a diagnosis. The HSDAG algorithm employs breadth-first search where the resolution of all minimal conflict sets leads to the identification of all minimal diagnoses. In our working example the diagnoses derived from the conflict sets CS_1 , CS_2 , and CS_3 are $DIAGS = \{\{c_7, c_8\}, \{c_7, c_{10}\}, \{c_8, c_9\}, \{c_8, c_{10}\}\}$.

The construction of such a HSDAG is exemplified in Figure 1. The HSDAG algorithm assumes the existence of a component that is able to detect minimal conflict sets. Our implementation is based on a version of the QUICKXPLAIN conflict detection algorithm introduced by [10]. Following a breadth-first search regime with the goal of identifying a minimal diagnosis, we have to resolve the conflict set CS_1 by checking whether c_7 or c_8 already represent a diagnosis. Both alternatives to resolve the conflict do not lead to a diagnosis since $(C_R - \{c_7\}) \cup C_{KB}$ as well as $(C_R - \{c_8\}) \cup C_{KB}$ are still inconsistent. We now can switch to the next level of the search tree since breadth first search inspects all nodes at level n of the search tree first and then extends the search to level $n+1$. Let us assume that the next conflict set returned by QUICKXPLAIN is $CS_2 = \{c_8, c_{10}\}$. Now, $(C_R - (\{c_7\} \cup \{c_8\})) \cup C_{KB}$ does not trigger further conflicts which means that $diag_1 = \{c_7, c_8\}$ has been identified as the first *minimal cardinality diagnosis*. Further details on the standard Hitting Set Directed Acyclic Graph (HSDAG) algorithm can be found in [14].

A major question to be answered is whether minimal cardinality diagnoses are leading to configurations of relevance, i.e., have a high probability of being selected by the user. We will provide answers in the following sections.

4 Calculating Personalized Diagnoses

As the number of possible diagnoses can become large, and presenting such a large number of alternatives to the user is inappropriate, we want to systematically reduce the number of alternatives with the goal to identify relevant diagnoses for the user and keep the diagnosis evaluation process as simple as possible. A simple heuristic to identify such diagnoses has already been presented in Section 3 where diagnoses have been ranked conform to their cardinality – in our working example $\{c_7, c_8\}$ has been identified as first minimal cardinality

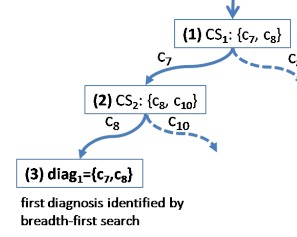


Figure 1. Hitting Set Directed Acyclic Graph (HSDAG) [14] for the working example. The first identified diagnosis is $diag_1 = \{c_7, c_8\}$. The algorithm returns minimal diagnoses with increasing cardinality, i.e., $diag_1 = \{c_7, c_8\}$ is a *minimal cardinality diagnosis*. The complete set of minimal diagnoses is $DIAGS = \{\{c_7, c_8\}, \{c_7, c_{10}\}, \{c_8, c_9\}, \{c_8, c_{10}\}\}$.

diagnosis. An alternative to this breadth-first search based approach is to exploit recommendation techniques [7] for the identification of *relevant diagnoses*, i.e., diagnoses which have a higher probability of being accepted by the user. In the following we will show how basic recommendation approaches can be exploited for the prediction of diagnoses which are relevant to the user. First we will show how we can determine diagnoses leading to configurations that are similar to the original set of user requirements (*similarity-based diagnosis selection*). Thereafter we will introduce a utility-based approach that uses preference data for guiding the HSDAG construction (*utility-based diagnosis selection*).

Similarity-based Diagnosis Selection. The idea of similarity-based diagnosis selection is to prefer those minimal diagnoses which lead to configurations resembling the original user requirements. In order to derive such diagnoses, we can exploit information contained in already existing configurations (see, e.g., the configuration log in Table 1). For each entry in Table 1 we can calculate its similarity with the user requirements in C_R . The similarity values of our working example are shown in Table 2. These values are calculated on the basis of the entries in Table 1 and preferences of our example user (see Table 3).²

Table 2. Similarity between configurations in Table 1 and the requirements C_R calculated on the basis of Formula 4: $simrec(C_R, conf_k)$, $k = 1..7$.

$conf_i$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$
similarity	0.45	0.60	0.43	0.25	0.30	0.36	0.14

Table 3. Importance values $w(c_i)$ specified by example user.

c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
8%	34%	8%	17%	8%	25%

The calculation of similarity values is based on three attribute-level similarity measures [11, 12, 19]. These measures calculate the similarity of a pair of attribute (a_i) of configuration $conf_k$ and the corresponding user requirement (c_i), for example, the similarity between attribute *ram* or configuration $conf_1$ and the user requirement c_9 ($ram \geq 2$) is 0.33

² Note that our approach does not rely on a specific preference elicitation method.

where we take the lower bound $ram=2$ as basis for similarity calculation. Depending on the characteristics of the attribute, one of the three measures is chosen: *More-Is-Better* (MIB), *Less-Is-Better* (LIB) or *Nearer-Is-Better* (NIB) [12]. For attributes like *harddisk size* or the *ram size*, the higher the value the better it is for the user (MIB). For attributes like *price*, the lower the value the more satisfied the user is (LIB). When the user specifies a certain type of CPU (no intrinsic value scale), we suppose the most similar is the preferred one. In those cases, the nearer-is-better (NIB) similarity measure is used.³

$$MIB : sim(c_i, a_i) = \frac{val(c_i) - min(a_i)}{max(a_i) - min(a_i)} \quad (1)$$

$$LIB : sim(c_i, a_i) = \frac{max(a_i) - val(c_i)}{max(a_i) - min(a_i)} \quad (2)$$

$$NIB : sim(c_i, a_i) = \begin{cases} 1 & \text{if } val(c_i) = val(a_i) \\ 0 & \text{else} \end{cases} \quad (3)$$

On the basis of the individual similarity values, Formula 4 calculates the overall similarity value between the sequence of user requirements (c) and the sequence of attribute values of configuration a . In this context $w(c_i)$ denotes the importance of requirement c_i for our example user.

$$simrec(c, a) = \sum_{i=1}^n sim(c_i, a_i) * w(c_i) \quad (4)$$

The similarity values shown in Table 2 will now be exploited for determining diagnoses in a personalized fashion (see Figure 2).

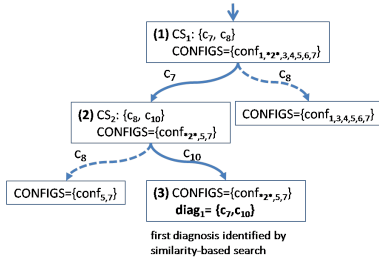


Figure 2. Similarity-based selection of diagnoses with PERSDIAG.

For the similarity-based selection of diagnoses we again assume that the QUICKXPLAIN algorithm [10] returns as first conflict set $CS_1 = \{c_7, c_8\}$. Now there are two possibilities of resolving CS_1 . If we delete c_7 from CS_1 , the following configurations $CONFIGS = \{conf_2, conf_5, conf_7\}$ are consistent with $\neg c_7$. This means that each of the configurations in $CONFIGS$ is *inconsistent* with the requirement c_7 and thus a potential candidate configuration for supporting diagnoses that include c_7 . If we delete c_8 from CS_1 , then $CONFIGS = \{conf_1, conf_3, conf_4, conf_5, conf_6, conf_7\}$. The configuration with the highest similarity compared to the original set of requirements $C_R = \{c_7, \dots, c_{12}\}$ is $conf_2$ contained in node (2) of

Figure 2. Consequently, node (2) of the HSDAG is further expanded which results in the next conflict set $CS_2 = \{c_8, c_{10}\}$ since $C_{KB} \cup (C_R - \{c_7\})$ is still inconsistent. With this expansion we have identified two alternative diagnoses, namely $\{c_7, c_8\}$ and $\{c_7, c_{10}\}$. The diagnosis $\{c_7, c_{10}\}$ will be rated higher since it is consistent with the configuration $conf_2$, the configuration with the highest similarity to the set of requirements, i.e., $conf_2 \cup C_{KB} \cup (C_R - \{c_7, c_{10}\})$ is consistent.

Utility-based Diagnosis Selection. The idea of utility-based diagnosis selection is to prefer those minimal diagnoses which preferably include requirements of low importance for the user. Following a utility-based approach [20] we are summing up the individual importance values (see Table 3) of the requirements part of a diagnosis in order to generate a corresponding ranking. The function $utility(C \subseteq C_R)$ returns a utility score for a specific set C which is a subset of the user requirements C_R (see Formula 5).

$$utility(C \subseteq C_R) = \frac{1}{\sum_{c_i \in C} w(c_i)} \quad (5)$$

For the utility-based selection of diagnoses we again assume that QUICKXPLAIN returns as first conflict set $CS_1 = \{c_7, c_8\}$ (see Figure 3). The importance value for c_7 is 0.08 whereas the importance value for requirement c_8 is 0.34 (see Table 3). By applying Formula 5 we derive the corresponding utility values, for example, $utility(\{c_7\}) = 1/0.08 = 12.5$ and $utility(\{c_8\}) = 1/0.34 = 2.9$. Since resolving the conflict set $\{c_7, c_8\}$ by deleting c_7 has a higher utility (application of Formula 5), the search for a diagnosis is continued with $C_R - c_7$ which results in the second conflict set returned by QUICKXPLAIN ($CS_2 = \{c_8, c_{10}\}$). Again, we sort the utility values for all nodes in the fringe of the search tree and come to the conclusion that extending the path $\{c_7, c_{10}\}$ is the best choice ($utility(\{c_7, c_{10}\}) = 4.0$). Since $(C_R - \{c_7, c_{10}\} \cup C_{KB})$ is consistent, $diag_1 = \{c_7, c_{10}\}$ is the first diagnosis identified (in this case the result is the same as the one determined by the similarity-based approach).

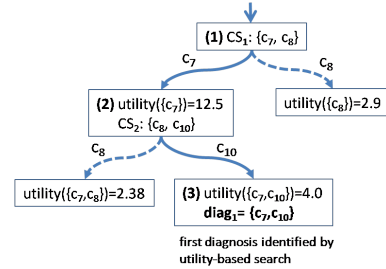


Figure 3. Utility-based selection of diagnoses with PERSDIAG.

Algorithm for Calculating Personalized Diagnoses. The algorithm for calculating best-first minimal diagnoses for inconsistent user requirements is the following (Algorithm 1 - PERSDIAG). We keep the description of the algorithm on a level of detail which has been used in the description of the HSDAG algorithm [14]. In PERSDIAG, the different paths of the HSDAG are represented as separate elements in a collection structure H which is initially empty. H stores all paths of the search tree in a best-first fashion, where the currently best path (h) is the one with the most promising (partial) diagnosis. If the theorem prover (TP) call $TP((C_R - h) \cup C_{KB})$ does not detect any further conflicts for the elements in h ($isEmpty(CS)$), a diagnosis is returned. The major role of the

³ For a detailed discussion of different types of similarity measures see, for example, [12, 19]. In the Formulas 1 – 3, $val(c_i)$ denotes the value of user requirement c_i , $min(a_i)$ denotes the minimal possible value of configuration attribute a_i , and $max(a_i)$ denotes the maximal possible value of attribute a_i .

theorem prover (*TP*) is to check whether there exists a configuration for C_R , disregarding the already resolved conflict set elements in h . If the theorem prover call $TP((C_R - h) \cup C_{KB})$ returns a non-empty conflict set CS , h is expanded to the paths containing exactly one element of CS each. In case that h is expanded, the original h must of course be removed from H ($delete(h, H)$). Afterwards, the new elements have to be inserted into H . This collection (H) is then finally sorted ($sort(H, k)$) according to the criteria defined in k .⁴ In this context, k represents the criteria used for selecting the next node to be expanded in the search tree which could be *breadth-first*, *similarity-based*, or *utility-based*.

Algorithm 1 PERSDIAG(C_R, C_{KB}, H, k)

```

{ $C_R$ : set of user requirements}
{ $C_{KB}$ : the configuration knowledge base}
{ $H$ : collection of all paths in the search tree (initially empty)}
{ $k$ : node evaluation criteria used by  $sort(H, k)$ }
{ $h$ : diagnosis returned}
 $h \leftarrow first(H)$ 
 $CS \leftarrow TP((C_R - h) \cup C_{KB})$ 
if isEmpty( $CS$ ) then
    return  $h$ 
else
    for all  $X$  in  $CS$  do
         $H \leftarrow H \cup h \cup X$ 
    end for
     $H \leftarrow delete(h, H)$ 
     $H \leftarrow sort(H, k)$ 
    PERSDIAG( $C_R, C_{KB}, H, k$ )
end if

```

5 Evaluation of Prediction Quality

To demonstrate the improvements achieved by our approach, we conducted an empirical study. Configuration data were gathered on the basis of an online user study conducted at the Graz University of Technology with 415 participants (82,4% male and 17,6% female) conform to the basic structure of Table 1. Each participant had to define his/her requirements (including the corresponding importance values – see Formula 4) regarding a predefined set of 12 computer attributes (*price, type of central processing unit, operating system, operating system language, amount of main memory, screen size, harddisk capacity, type of DVD drive, web cam, type of graphic card, amount of graphic card memory, and type of service*). After this requirements specification phase participants were informed about the fact that for the specified set of requirements no solution could be found (the goal was to confront each participant with such a situation). The system then presented a list of max. 50 alternative configurations (repair configurations inconsistent with the current set of requirements) which have been calculated by a computer configuration knowledge base built for the product set offered by a commercial website.⁵ The ordering of the configurations in this list was randomized and the participants were enabled to navigate in the list and to order the configurations regarding

different criteria such as the price (less is better), the size of the harddisk (more is better), or the number of fulfilled requirements (more is better). The participants then had the task to select one out of the presented repair configurations that appeared to be the most acceptable one for them.

Based on the data collected in the user study we evaluated the three presented approaches w.r.t. their capability of predicting diagnoses that are acceptable for the user. The first approach is based on the algorithm proposed by [14], where diagnoses are ranked according to their *cardinality* and diagnoses of the same cardinality are ranked according to their calculation order (see Section 3). The second approach identifies personalized diagnoses on the basis of a *similarity*-based node expansion strategy in HSDAG construction (see Section 4). The third approach uses a *utility* measure to find relevant diagnoses for the user (see Section 4). Due to the fact that no solution was made available for the original set of requirements, for each such set we could determine a set of diagnoses that indicated which of the requirements had to be relaxed or adapted in order to be able to identify a solution.

We were then interested in the *prediction accuracy* of the three different diagnosis approaches (*cardinality-based*, *similarity-based*, and *utility-based*). First, we analyzed the distance between the *predicted position* of diagnoses leading to a selected repair proposal and their *expected position* (which is 1). We measured this distance in terms of the *root mean square deviation* – *RMSD* (see Formula 6). The results of this analysis are shown in Table 4. We can see that the utility-based diagnosis approach has the lowest RMSD which is 0.97. The similarity-based approach shows a similar RMSD value (1.03), and the cardinality-based approach shows the worst performance (RMSD = 1.64).

$$RMSD = \sqrt{\frac{1}{n} \sum_{i=1}^n (\text{predicted position} - \text{expected position})^2} \quad (6)$$

Table 4. Root Mean Square Deviation (RMSD) values.

Cardinality-based	Similarity-based	Utility-based
1.64	1.03	0.97

Although RMSD is a good quality estimate it provides only limited information about the precision of the prediction. Therefore we analyzed the precision of the three diagnosis approaches – the precision measure is shown in Formula 7. The basic idea is to provide a measure on how often a diagnosis that leads to the repair configuration selected by the participant is among the top-n ranked diagnoses. As can be seen in Table 5, the utility-based approach has the highest prediction accuracy in terms of precision, followed by the similarity-based diagnosis approach. The cardinality-based approach has the worst performance in terms of prediction accuracy.

We were interested whether we could detect a statistically significant difference between the three diagnosis approaches in terms of prediction accuracy. Therefore we conducted a pair wise comparison between the diagnosis approaches on the basis of a Mann-Whitney-U-Test. We could detect a significant difference between the prediction accuracy of *utility-based* diagnosis and *cardinality-based* diagnosis ($p = 5.69e^{-9}$) and between *similarity-based* and *cardinality-based* diagnosis ($p < 2.2e^{-16}$). There was no significant difference between

⁴ Note that the HSDAG pruning is implemented by the functionalities of $sort(H, k)$.

⁵ The knowledge base has been implemented for the 50 configurations extracted from www.dell.at. We chose this simple knowledge base in order to avoid biases, for example, in terms of presenting only solutions that are near the original set of requirements.

utility-based and similarity-based diagnosis in terms of prediction accuracy ($p = 0.5952$).

$$\text{precision} = \frac{|\text{correctly predicted diagnoses}|}{|\text{predicted diagnoses}|} \quad (7)$$

Table 5. Precision of the three diagnosis approaches.

	top-1	top-2	top-3
Cardinality-based	0.51	0.75	0.87
Similarity-based	0.70	0.87	0.97
Utility-based	0.74	0.89	0.96

6 Related Work

Model-based Diagnosis. The increasing size and complexity of configuration knowledge bases motivated the application of model-based diagnosis [2, 14] for testing and debugging purposes [6]. Similar reasons led to the application of model-based diagnosis in technical domains such as hardware designs [3] and onboard diagnosis for automotive systems [16]. The work presented in [6] has a special relationship to the concepts presented in this paper: [6] focus on the application of model-based diagnosis to the identification of faults in configuration knowledge bases where test cases are used to induce conflicts in a given configuration knowledge base. In addition, a first approach to calculate diagnoses for inconsistent user requirements is presented which is based on breadth-first based HSDAG construction. In this paper we have shown how to apply basic recommendation algorithms (similarity-based and utility-based) to improve the diagnosis algorithms in terms of prediction accuracy.

Diagnosing Inconsistent Requirements. An approach to suggest personalized repair actions for inconsistent requirements in the context of knowledge-based recommendation tasks has been introduced by [8]. The underlying idea is to apply the concepts of Model-based Diagnosis (MBD) [2, 14] to determine change proposals for minimal cardinality sets of inconsistent requirements in the case of a given pre-defined list of products. In [13] such minimal cardinality sets are denoted as minimal exclusion sets. In case-based recommendation scenarios [9, 12] the complement of a minimal exclusion set is denoted as maximally successful sub-query. The concept of representative explanations has been introduced by [13]. Representative explanations follow the idea of generating diversity in sets of diagnoses (minimal exclusion sets). The approach does not explicitly take into account the preference structure of the current user but rather tries to determine diagnosis sets that satisfy the requirement that each element part of at least one diagnosis is also contained in at least one of the diagnoses presented to the user.

7 Conclusion

In this paper we introduced an algorithm (PERSDIAG) for the determination of personalized diagnoses. The algorithm significantly improves the prediction quality compared to state of the art diagnosis approaches. PERSDIAG follows a best-first search regime and can be parameterized with different kinds of selection strategies regarding the expansion of the search tree. We have compared different expansion strategies (cardinality-based, similarity-based, and utility-based) within the scope of an empirical study. The results of this study show

the advantages of personalized diagnosis calculation compared to existing breadth-first based search in terms of prediction quality. These results provide a solid basis for improving existing industrial applications regarding the determination of diagnoses for inconsistent requirements.

References

- [1] R. Burke, *Knowledge-based recommender systems*, Library and Information Systems, 69(32):180–200, 2000.
- [2] J. de Kleer, A. Mackworth and R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.
- [3] G. Friedrich, M. Stumptner, and F. Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(2):3–39, 1999.
- [4] G. Friedrich, G. Gottlob, and W. Nejdl. Physical Impossibility Instead of Fault Models. *Proceedings of the 8th National Conference on Artificial Intelligence AAAI/IAAI’90*, pp. 331–336, Boston, MA, 1990.
- [5] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker. Configuration Knowledge Representations for Semantic Web Applications. *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing (AIEDAM)*, 17(2):31–50, 2003.
- [6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.
- [7] A. Felfernig, G. Friedrich, and L. Schmidt-Thieme. Introduction to the IEEE Intelligent Systems Special Issue: Recommender Systems. *IEEE Intelligent Systems*, 22(3):18–21, 2007.
- [8] A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible repairs for inconsistent requirements. *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 791–796, Pasadena, CA, 2009.
- [9] P. Godfrey. Minimization in cooperative response to failing database queries. *International Journal of Cooperative Information Systems*, 6(2): 95–149, 1997.
- [10] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. *Proceedings of the 19th National Conference on Artificial Intelligence, AAAI/IAAI’04*, pp. 167–172, San Jose, CA, 2004.
- [11] 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.
- [12] D. McSherry. Maximally Successful Relaxations of Unsuccessful Queries. *15th Conference on Artificial Intelligence and Cognitive Science*, pp. 127–136, Galway, Ireland, 2004.
- [13] B. O’Sullivan, A. Papadopoulos, B. Faltings, and P. Pu. Representative explanations for over-constrained problems. *Proceedings of the 22nd National Conference on Artificial Intelligence, AAAI/IAAI’07*, pp. 323–328, Vancouver, Canada, 2007.
- [14] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
- [15] D. Sabin and R. Weigel. Product Configuration Frameworks - A Survey. *IEEE Intelligent Systems, Special Issue on Configuration*, 13(4):42–49, 1998.
- [16] M. Sachenbacher, P. Struss, and C. Carlen. Prototype for Model-Based On-Board Diagnosis of Automotive Systems. *AI Communications*, 13(2):83–97, 2000.
- [17] C. Sinz and A. Haag. Configuration. *IEEE Intelligent Systems, Special Issue on Configuration*, 22(1):78–90, 2007.
- [18] E. Tsang. Foundations of Constraint Satisfaction. *Academic Press, London and San Diego*, 1993.
- [19] D. Wilson and T. Martinez. Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.
- [20] D. Winterfeldt and W. Edwards. Decision Analysis and Behavioral Research. *Cambridge University Press*, Cambridge, England, 1986.

Constraints filtering and evolutionary algorithm for interactive configuration and planning

P. Pitiot¹, E. Vareilles¹, M. Aldanondo¹, M. Djefel^{1,2}, and Paul Gaborit¹

Abstract. This communication aims to associate the product configuration task with the planning of its production process in order to make consistent decisions while trying to minimize cost and cycle time. A two step approach is described with relevant aiding tools. During the first one, configuration and planning are considered as two constraint satisfaction problems and are interactively assisted by constraint propagation. The second one, thanks to a multi-criteria optimisation relying on a constrained evolutionary algorithm, proposes a set of solutions belonging to a Pareto front minimizing cost and cycle time to the user. After a problem introduction and a global description of the integrated support tool, the paper focuses on the optimisation process with interesting quantified results.

1 INTRODUCTION

This paper presents an integrated support tool which at first allows interactive configuration of a product and interactive planning and scheduling of its production process, and then minimizes conflicting criteria cost and cycle time. The configuration of a private aircraft will be used as an example to illustrate our research work.

In literature, most of the research into product configuration and production planning treats them independently. However, the decisions of product configuration obviously have strong consequences on the planning of its production process (for example, a luxury finish requires at least two additional months. On the other hand, planning decisions can provide hard constraints to product configuration (for example, such assembly duration forbids the use of such a kind of engine). Therefore, we propose to associate these two problems so that (i) the consequences of each decision of product configuration can be propagated toward the planning of its production process and (ii) the consequences of each process planning or scheduling decision can be propagated towards the product configuration. As we target interactive assistance in order to allow some kind of “what if” operating mode, we need to be able to show the consequences of each user’s elementary requirement. A user’s elementary requirement can be defined as a restriction of the domain of a variable involved in configuration (for example, number of seats belongs to [6, 12]) or in planning (for example, due date is prior to 31/10/2010). We consequently do not intend to process all the requirements simultaneously in a single shot to get a solution for both problems but rather to progressively lead the user to a solution for both product configuration and planning of its production.

In the field of configuration, many authors, among whom [1] or [2], showed that product configuration can be efficiently modeled and aided when it is considered as a Constraints Satisfaction Problem (CSP).

In a same way, authors interested in planning and scheduling as [3] or [4] have shown that these problems could be also modeled and aided when considered as a CSP. We therefore propose to consider configuration and planning problems as two constraint satisfaction problems. We assume that a constraint based model of a generic product and the same kind of model for a generic production plan can be established and we restrict configuration and planning tasks to the instantiation of these two models. We also limit the scope of this paper to infinite capacity planning. To support interactive assistance, we only use the filtering or constraint propagation capabilities of the CSP framework. We finally link the two problems (configuration and planning) and the coupling constraints proposed in [5] together to propagate the consequences in both directions.

In the previous system, a product can be entirely configured and its production process entirely planned. “Entirely” means to restrict the solution space to a single solution, each problem variable having a single value. But we are not interested in this operating mode. We assume that it is possible to decompose the set of user’s requirements in two sub-sets: non-negotiable requirements and negotiable ones. Our idea is to process interactive configuration and planning with the first sub-set or requirements only (non negotiable) and achieve a first reduction of the solution space. Remaining variable affectations (remaining solution space) are kept for multi-criteria considerations in the second step of our proposition.

In most industrial cases, the resulting products configured are characterized by criteria such as performance and product cost while relevant production plans are associated with cycle time and production cost. A solution is always a compromise of somehow contradictory criteria. In this presentation we only consider two criteria: cost (product cost and production cost) and cycle time for production planning. Hence, the next step is to find solutions that belong to the Pareto front (time/cost) among the solution space restricted during the first step. Multi-criteria optimization techniques and more accurately Evolutionary Algorithm (EA) (see [6] and [7]) have the advantage to avoid the aggregation of criteria and can provide solutions on a Pareto front in a rather simple way. Thanks to an evolutionary approach, the second step will perform a second reduction of the solution space for both product and plan and provide solutions on the Pareto front. Finally, the user can finish the process by selecting the solution that fits his specific time/cost compromise. The next section describes our proposition for the first step with an example, then our proposition for the second step with an evolutionary approach is detailed.

2 CONFIGURATION AND PLANNING MODELS AND CONSTRAINT PROCESSING

The configuration model (left part of figure 1) gathers product descriptive variables (for example: aircraft range, number of engines, type of finish...) and product cost variables (finish cost, engine cost...) that are either symbols or discrete numbers. Configuration constraints (for example black solid lines that can link aircraft range

¹ Toulouse University – Mines Albi – Centre de Génie Industriel, France, {pitiot, aldanondo, vareilles, djefel, gaborit}@enstimac.fr}

² Toulouse University – INSAT – Laboratoire Toulousain de Technologie et d’Ingénierie des Systèmes, France, mdjefel@lattis.fr

and engine type together) and cost definition constraints (for example grey solid lines that can link engine type and number of engines with engine cost) correspond most of the time with discrete tables showing allowed combinations of allowed values. In this discrete problem, the associated CSP is discrete and the filtering provided by arc consistency technique [8] allows interactive configuration and cost estimation.

The planning model (right part of figure 1) gathers a set of planning operations (like manufacturing, assembling...) linked with ordering constraints. Each operation is defined with three operation temporal variables (starting date, ending date, possible duration) and eventually resource variables (required resource, quantity of required resource). We assume that the three temporal variables are real variables defined with intervals while resource type is symbolic and resource quantity a real variable. The cost of an operation is a real variable that depends on the resource type or quantity and the operation duration. As we consider planning with infinite capacity of resource, the constraints are as follows. Ordering constraints between operations (if task Y is after task X then starting date of Y is greater than or equal to ending date of X) and operation duration constraints (ending date equals starting date plus possible duration) are numerical constraints (black solid lines). The constraints that link possible duration with required resource and/or quantity of required resource and/or cost are mixed constraints (black and grey solid lines). Our numerical constraints are simple calculations (+, -, *, /, =, >, <), therefore they respect the hypothesis of bound Consistency proposed by Lhomme in [9]. Based on interval arithmetic, bound consistency is gathered with arc consistency and allows interactive planning and scheduling.

A mixed constraint between product description variables and planning operation variables (black dotted lines) allows to propagate decision consequences from configuration to planning and from planning to configuration. [5] can be referred for more details. The first global criterion, i.e. total cost, is calculated thanks to a numerical constraint as the sum of product cost and operation cost (grey dotted lines of figure 1). The second one, the production cycle time corresponds with the ending date of the last operation.

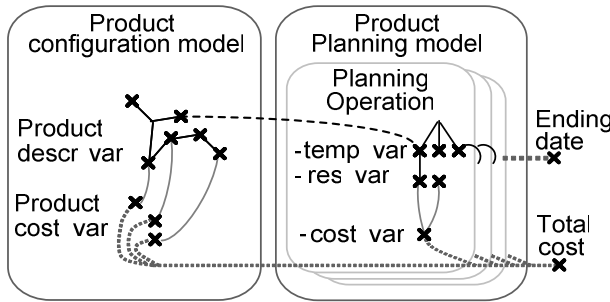


Figure 1. Configuration and planning model.

3 OPTIMIZATION PROBLEM TO ASSIGN

3.1 Definition of the optimization problem

The first interactive filtering step leads to the restriction of the initial feasible space (noted (a) on figure 2) to a restrained area (noted (b) on figure 2). This corresponds to the filtering of a customer's non-negotiable requirements. The filtering system provides domain bounds for every criteria variable (minimal and maximal values for cost and cycle time). The restrained area contains

solutions corresponding to different remaining decisions to fulfil according to the remaining requirement. Notice that this area also contains unfeasible space where there is no solution due to the constraints of the problem to solve. The aim of the optimization process is to find a selection of the closest solutions to the Optimal Pareto front. Various metaheuristics may be used to solve this problem. In this research work, we focus on evolutionary algorithms for their ability to propose multiple solutions while solving a multiobjective problem. But classical evolutionary algorithms have to be adapted to take into account the constraints of the problem.

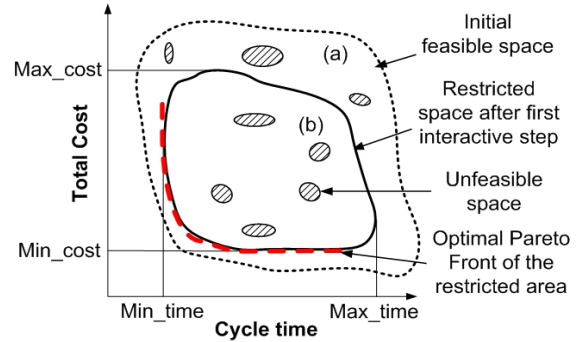


Figure 2. Illustration of the reduced search space to optimize (b) from the initial feasible space (a) after the filtering of non-negotiable requirements.

3.2 Formal definition of optimization problem

The constrained optimization problem (O-CSP) is defined by the quadruplet $\langle V, D, C, f \rangle$, where V is the set of decision variables, D the set of domains linked to each variable of V , C the set of constraints on variable of V and f the multi-valuated fitness function. Here the aim is to minimize both cost and cycle time. The set V gathers: the product descriptive variables and the resources variables (required resource and choice or quantity of required resource). Constraints, as seen in section 2, link the variables of V with all the variables of the two problems. The filtering system allows dynamically computing modification on the domain of variables. This OCSF is a difficult problem to solve. The existing methods to handle constraints in EA are often computationally expensive. The next section briefly presents existing approaches.

3.3 Overview of constrained optimisation approaches

Initially, EA deal with large combinative unconstrained problem. They are used to solve multiple problems like project planning or project configuration. But real world problems are generally constrained. Many research studies try to integrate constraints in EA. C. Coello Coello comprehensively surveys the state of the art of these methods [10] (more than eight hundred references). It seems that classical EA performance decreases while the number of constraints increases. Constraints imply feasible and un-feasible areas in search space. Then the ratio between the size of feasible and un-feasible space guides the choice of specific methods to handle constraints in EA [11]. Four kinds of methods deal with this problem: penalty functions [12], repair methods [13], approaches that separate objectives and constraints (Multiobjective Optimization (MO) techniques) [14] and specific representations or operators [15], [16].

The first one, penalty functions, is the most common way to integrate violation of constraints in the objective function. For each individual, the level of violation of constraint is added to the fitness function during the evaluation phase. The aim is to increase the probability of selection for feasible or near feasible solutions. The main drawback of such an approach is that the boundary between feasible and unfeasible regions is usually difficult to grasp. Furthermore, it requires the definition of the weights needed to aggregate the violation of different constraints. The repair methods only try to deal with feasible individuals. As soon as an unfeasible individual is generated, a specific operator redirects it towards the feasible space. The difficulty is thus to elaborate a performing repairing algorithm that preserves the diversity of individuals. The same problem appears with MO approaches. These ones integrate the satisfaction of each constraint (or a group of constraints) as a specific objective.

Finally, the specific operators or representations approaches aim at preserving the feasibility of the individuals during their construction. Kowalczyk previously proposed in [15] the use of constraint consistency to prevent variable instantiations which are not consistent with the constraints of the problem. However, his study suggests that the supplementary computing time needed for constraint propagation may be very expensive in comparison with optimization process.

In this paper, we will focus on specific evolutionary operators that prune search space using constraint filtering. The arc-consistency is less time consuming than other CSP solving systems, but it does not guarantee the global coherence of the model.

3.4 The approach proposed

3.4.1 Overview of modified EA used

EA used is adapted from the SPEA2 method [17] with classical evolutionary steps (initialisation, fitness assignment, selection, stopping criterion and evolutionary perturbation operators) illustrated on figure 3. It is one of the most useful Pareto-based methods founded on the preservation of a selection of best solutions in a separate archive (archive on generation t is noted A_t and population P_t). It ensures both a good convergence speed and the preservation of diversity of solutions. Zitzler shows that the maximal complexity of the overall algorithm is $O(N_{pop}^3)$ with N_{pop} the number of solutions in current population.

Five parameters are required: Archive (N_{arch}) and size of population (N_{pop}) used to control the diversity and convergence properties according to the problem to solve, a number of generation (T) used as a stopping criterion and probabilities for crossover (P_{cross}) or mutation (P_{mut}) operators (probabilities of selection of individuals for each operator). The classical initialisation and evolutionary operators steps (framed with a bold line on figure 3) have been modified to interact with a filtering system in order to build or modify the individuals that satisfy the constraints of the problem.

SPEA2 includes a performing evaluation strategy that brings a well-balanced density of population on each area of search space. The calculation of fitness, given by equation (2), is based on the Pareto dominance between individuals (noted $\succ$) and the measurement of solution density. It firstly requires the calculation of the strength of each individual of the population and the archive. The strength $S(i)$ of an individual i , given by the equation (1), represents the number of individuals it dominates.

$$S(i) = |\{j | j \in P_t + A_t \wedge i \succ j\}| \quad (1)$$

$$F(i) = \sum_{j \in P_t + A_t, j \succ i} S(j) + \frac{1}{\sigma_i^k + 2} \quad (2)$$

with σ_i^k the Euclidian distance to the k^{th} nearest neighbour and $k = \sqrt{N_{pop} + N_{arch}}$

The first term in the equation (2) represents the strength of individuals j that dominate i . For a Pareto solution, it is equal to zero. So here, the aim is to minimize $F(i)$. The second term, which represents the information on density, allows to discriminate the individuals with identical scores according to the density of individuals in the surrounding area.

After the fitness assignment, the step of environmental selection updates individuals of the archive. Firstly, every Pareto individual of the population and the previous archive is copied to the archive. Then, if there are too many solutions in the archive, an archive truncation process that preserves the boundary solutions is run. Otherwise, if there are not enough individuals, the archive is filled with a selection of best remaining individuals of the previous population and archive (dominated ones).

Finally, if the number of generations done is inferior to the number given by the user as a stopping criterion, the mating selection step selects N_{pop} individuals in the archive to constitute the mating pool. This is done by a binary tournament between individuals of the archive. This mating pool will be used by crossover and mutation operators, in order to build the next generation.

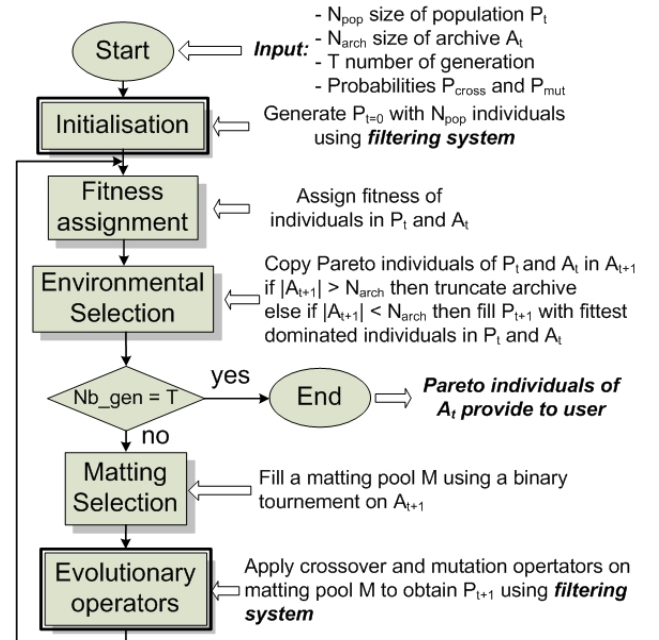


Figure 3. SPEA2 algorithm [17] adapted to use filtering system

The following sections precise the initialisation, crossover and mutation operators that interact with the filtering system to build or modify the individuals according to the constraints of the problem (e.g. to keep them in the feasible space).

3.4.2 Initialisation operator

Thanks to this operator, a well-diversified set of initial individuals is obtained. The pseudo-algorithm below precises its general behaviour. For each individual to create, every gene (decision variable) is randomly instantiated into its current domain. In order to avoid the generation of unfeasible individuals, the domain of every remaining gene is dynamically updated by filtering after each instantiation. If an individual is incoherent, a limited backtrack process cancels one of the previous choice, then the individual is filtered again until the values of the remaining variable are consistent with the constraints of the problem. If the backtrack limit is reached, the individual is abandoned to bound the number of backward step and thus the computational time spent by the filtering. This process (random instantiation then filtering) is repeated until all the genes of every individual are instantiated.

Pseudo-algorithm of initialisation operator:

Input: Empty initial population P_0
Size of population N_{pop}

Start
Create a population of empty individuals
While (all individuals aren't finished)
 Select an individual not finished
 Select an uninstantiated variable on it
 Select a state in variable domain
 Ask for a filtering of individual
 If (coherent instantiation) **do**
 Update domains of remaining genes
 Else
 while (individual is unfeasible)
 If (Backtrack counter limit reach) **do**
 Restart with empty individual
 Else
 Randomly select an instantiated variable
 Uninstantiated it
 Increase Backtrack counter
 Ask for a filtering of individual
 EndIf
 EndWhile
 EndIf
EndWhile
End

3.4.3 Uniform mutation operator

This operator introduces a random perturbation on the evolutionary process that allows escaping from sub-optimal areas and thus the exploration of search space. It modifies the instantiation of some genes on individuals selected according to the mutation probability P_{mut} among mating pool M_t . Every individual and their genes selected for the mutation are beforehand chosen randomly. The individuals are selected in the mating pool, then selected genes are un-instantiated. The filtering system updates the domain of these variables according to the instantiation of others genes. Finally, the mutation of the selected genes itself is achieved in the same way as during the initialisation (instantiation, filtering and backtrack limit). When every gene is instantiated, the mutation process end and the individuals are added to the next generation P_t .

3.4.4 Uniform crossover operator

This operator allows to shuffle randomly and uniformly the genes of two individuals (named parents) selected in the mating pool M according to the crossover probability P_{cross} . The pseudo-algorithm on the next page precises its general behaviour. To achieve this task, a crossover table (represented on the upper part of figure 4) is filled. It allows to select which genes will be exchanged between parents. The crossover corresponds to a selected way on a binary tree (represented on the lower part of figure 4), where each branch is linked to the crossover of a particular gene. An instantiation of the crossover table (filling of lines that corresponds to child) is equivalent to the selection of a path on the crossover tree.

Firstly, the two first lines of the table are randomly filled (the first line corresponds to the order of treatment of genes, and the second one selects which genes will be crossed). Notice that the order of treatment of a specific gene is chosen randomly to avoid the dominance of genes by their position in the chromosome. The initialisation of crossover flags corresponds to a random selection of a path in the graph.

During the crossover operation, the system tries to achieve this random crossover. At every gene instantiation, the filtering system updates the domain of remaining genes. The crossover table is initially filled identically for both children, but if an individual become unfeasible, a specific backtrack is done by changing some crossover flag in the table. A supplementary flag (third line) is added to the table that memorizes the unfeasible way on the tree in case of backtrack. Then a backtrack counter limits the number of backward steps. If the backtrack limit is reached, the corresponding child is abandoned. Finally, every feasible child is added to the next generation.

Gene number	2	5	1	6	3	4
Crossover flag	0	1	1	0	0	1
Memory flag	0	0	0	0	0	0
Parent 1	1	3	2	5	1	1
Parent 2	2	1	3	5	2	1
Child 1	1					
Child 2	2					

1st step → Filling sens →

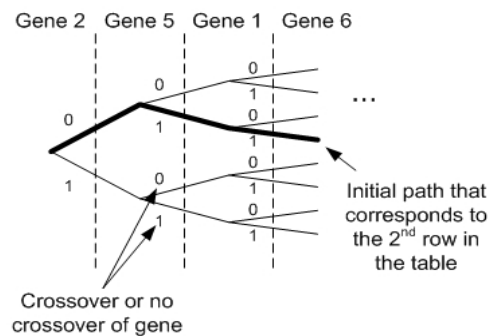


Figure 4. Crossover table filling and corresponding crossover tree

Pseudo-algorithm of crossover operator:

```

Input:      Pcross Crossover probability
              M mating pool with Npop individuals

Start
For(all individuals in M)
  If(Random(0,1) < Pcross)do
    Randomly select an other Parent on M
    Create corresponding crossover table
    Randomly fill 1st row with gene number
    Randomly fill 2nd line with 0 or 1
    Filled 3rd line with 0
    While(crossover done for both children)do
      Apply next gene crossover according to
      2nd line (1 crossover / 0 no crossover)
      Ask for a filtering of both individuals
      If(a child is unfeasible)do
        If(Backtrack counter limit reach)do
          Abandon corresponding child
        Else
          While(memory flag of current gene = 1)do
            Set memory flag of current gene to 0
            Go back to previous gene
          EndWhile
          Set memory flag of current gene to 1
          Switch crossover flag of current gene
        EndIf
      EndIf
    EndWhile
  EndIf
EndFor
End

```

4 PRELIMINARY RESULTS

We have performed various tests on aircraft problem¹ (42 variables among whom ten are taken into account in EA, 42 constraints, 2.799.360 possible solutions (e.g. different possible chromosomes), among 1.098.360 are feasible solutions) to evaluate the proposed approach. The initial search space shape (space of existing solutions for the whole problem without any restriction done by the user) is represented on figure 4 (smallest dark points). The first step of our method suggests that the user defines its non-negotiable requirements by selecting the required values for some decision and asks for a filtering of the model. For example, the user can specify a global size of the aircraft (here, he needs a medium aircraft that can contain six or eight seats) and a specific finish (here, he does not want standard or custom finish). The grey points on figure 3 represent the search space shape of the reduced model. The reduced model has 349.920 possible chromosomes and 218.700 feasible solutions. Notice that the user's requirements exclude the most expensive and the most economical solutions.

Various evolutionary settings were investigated in each model (initial and reduced ones). In most runs, the modified EA was able to provide very good solutions, near Pareto-optimal solutions. Figure 5 presents the Pareto-optimal front and the archive found after one run of the proposed EA (25 generations, population size: 20, archive size: 40, crossover rate: 0.8, mutation rate: 0.4) on the reduced

problem. On average for twenty runs, 97.9% of the optimal Pareto front is reached by the modified EA.

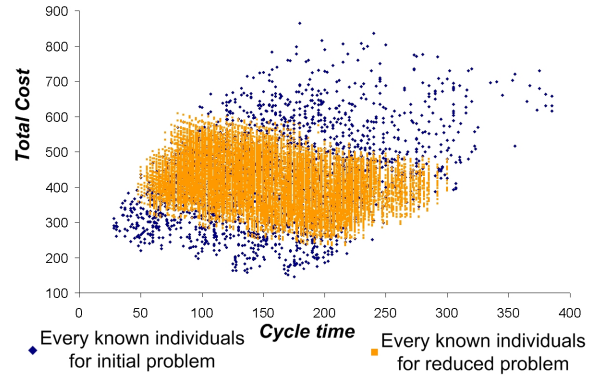


Figure 4. Search space shape for initial and reduced aircraft model. Those points were obtained using a random sampling (ten thousands individuals for each model).

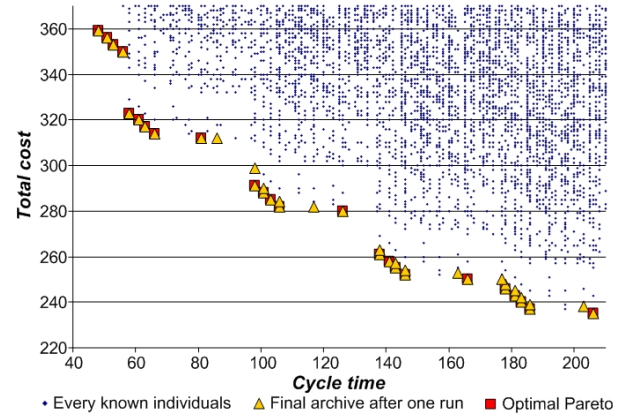


Figure 5. Detail of archive resulting from one run of modified EA (triangles), best known Pareto Front obtained by EA stopped after hundred generations (squares) and every individual founded for the reduced problem (dark points).

To evaluate the algorithm results (convergence speed and diversity of solutions), we use the hypervolume metric defined by Zitzler and Thiele in [18] and illustrated on figure 6. This metric both evaluates convergence and diversity proprieties. Its formula, given by equation (1), corresponds to the union of domination areas for each Pareto solution in the archive (the domination area of a solution is the portion of space dominated by the solution).

$$HV = \bigcup_{i=1}^{NP_{arch}} C_i \quad (3)$$

where NP_{arch} is the number of Pareto solutions in archive and C_i is the hypercube that links solution i to the reference point W (point that corresponds to the worst values for each objective (noted Max on figures 2 and 6)). Here these values are given by the filtering system during initialization. The fittest and most diversified set of solutions is the one that maximizes hypervolume.

Figure 7 presents the evolution of the hypervolume achieved with a modified EA on a reduced model (average of 20 runs). The area covered by the solutions of the Pareto front increases quickly in the first generations, and is very close to the optimal one ($HV_m=182149$

¹ Aircraft model could be investigated on-line: select model Aircraft at: <http://cofiade.enstimac.fr/cgi-bin/cofiade.pl>

with a relative standard deviation of 0.218% for an optimal hypervolume $HV_{opt}=182304$ on final generations. Among twenty runs, fifteen find the optimal Pareto front with an average computing time of 1552 seconds (about 26 minutes). On the initial model and with identical settings, EA achieved similar results: $HV_m=360029$ with a relative standard deviation of 0.22% and an average computing time of 5380 seconds ($HV_{opt}=360432$ for the initial model). These results are very interesting, because it clearly shows that in a reasonable amount of time, it is possible to propose a set of solutions that permits the user to decide about his own cost/cycle time compromise. This is much more accurate than solutions frequently proposed in industrial configuration software which are based on the selection of default values that try to minimize either cost or cycle time.

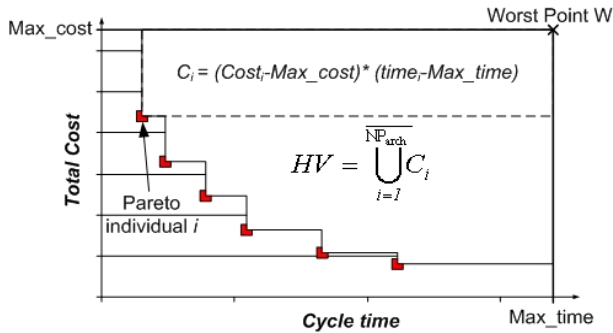


Figure 6. Example of hypervolume link to a Pareto front.

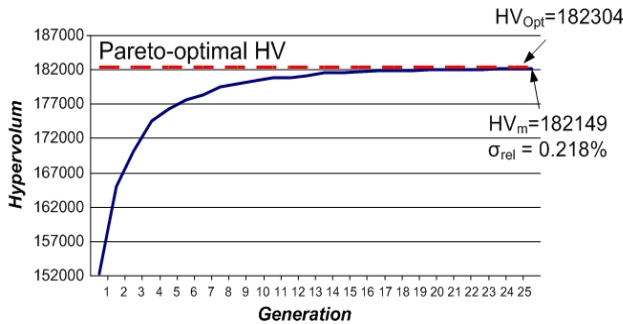


Figure 7. Evolution of hypervolume obtained on a reduced problem (average of twenty runs with following EA settings: 25 generations, population size: 20, archive size: 40, crossover rate: 0.8, mutation rate: 0.4). The dotted line represents the optimal value computed with long EA runs.

5 CONCLUSION

In this paper, we present an efficient integrated support tool for coupled product configuration and project planning, using an interactive constraint filtering system then a modified evolutionary optimization system. The modified EA is based on the interaction with the filtering system that prunes the search space and thus reduces the search effort by limiting it to the feasible individuals. Standard evolutionary operators are adapted to take advantage of the filtering. First experiments indicate that this method is well adapted for an interactive integrated support tool. It generates near-optimal Pareto-solutions in a reasonable computing time. It allows the

user to decide efficiently about his cost/cycle-time compromise when dealing simultaneously with configuration and planning.

ACKNOWLEDGMENT

The authors would like to thank the French National Research Agency (ANR), the Aerospace Valley for their involvement, their partners in the ATLAS project: Pulsar Innovations, Sigma-Plus and LAAS-CNRS, LGP-ENIT, IMS-LAPS and ESTIA from France.

REFERENCES

- [1] T. Soininen, T. Tiihonen, T. Männistö and R. Sulonen, "Towards a General Ontology of Configuration", in *AIEDAM*, vol. 12 - n° 4, pp 357-372, 1998.
- [2] U. Junker, *Chap 26 of Handbook of Constraint Programming*, Elsevier, 2006.
- [3] R. Dechter, I. Meiri and J. Pearl, "Temporal Constraint Satisfaction Problems", *Artificial Intelligence*, n°49, pp 61-95, 1991.
- [4] R. Barták, M. Salido and F. Rossi, "Constraint satisfaction techniques in planning and scheduling", *Journal of Intelligent Manufacturing*, vol. 21, issues 5-15, 2010.
- [5] M. Aldanondo, E. Vareilles, M. Djefel, C. Baron, "Product and Process Configuration: A constraint based approach", *proc. of IEEM'08*, Singapore, vol 1-3, p. 2010-2014, 2008.
- [6] B. Li, L. Chen, Z. Huang and Y. Zhong, "Product configuration optimization using a multiobjective GA", *I.J. of Adv. Manufacturing Technology*, vol. 30, p. 20-29, 2006.
- [7] R. Chelouah, C. Baron, M. Zholghadri and C. Gutierrez, "Meta-heuristics for System Design Engineering", *Studies in Computational Intelligence*, vol. 203, p. 387-423, 2009.
- [8] C. Bessiere, *Chap 3 of Handbook of Constraint Programming*, Elsevier, 2006.
- [9] O. Lhomme, "Consistency techniques for numerical CSPs". *IJCAI 93*, Chambéry, France, p 232-238, 1993.
- [10] C. Coello Coello, Computer science department: CINVESTAV, www.cs.cinvestav.mx/~constraint/.
- [11] C. Coello Coello, "Theoretical and numerical constraint-handling techniques used with EAs : A survey of the state of art", *Computer Methods in Applied Mechanics and Engineering*, vol. 191, n°11-12, p. 1245-1287, 2002.
- [12] J.T. Richardson, M.R. Palmer, G. Liepins and M. Hilliard, "Some guidelines for GA with penalty functions", J.D. Schaffer editor, *3rd int. conf. on G.A.*, p. 191-197, 1989.
- [13] S. Salcedo-Sanz, "A survey of repair methods used as constraint handling techniques in evolutionary algorithms", *Computer science review*, p. 175-192, 2009.
- [14] L. Clevenger, L. Ferguson and W.E. Hart, "Filter-based Evolutionary Algorithm for Constrained Optimization", *Evolutionary Computation*, vol. 13, issue 3, p. 329-352, 2005.
- [15] R. Kowalczyk, "Constraint Consistent Genetic Algorithms", *Proc. of IEEE conf. on evolutionary computation*, p. 343-348, 1997.
- [16] Z. Michalewicz and G. Nazhiyath, "Genocop III: A co-evolutionary algorithm for numerical optimization with non linear constraints", Fogel editor, *Proc. of the 2nd IEEE conf on evolutionary computation*, p. 647-651, 1995.
- [17] E. Zitzler, M. Laumanns and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm", *Technical Report 103*, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, 2001.
- [18] E. Zitzler and L. Thiele "Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study", in A.E. Eiben, T. Bäck, M. Schoenauer, H.P. Schwefel editors, *Proc. of 5th Conference on Parallel Problem Solving from Nature (PPSN V)*, Springer Verlag, p. 292-301, 1998.

A Flexible Approach to Product Configuration exploiting Fuzzy Knowledge and SQL

Luigi Portinale and Massimiliano Bazzani¹

Abstract. Flexible retrieval of configurable products is one of the most challenging task for e-commerce applications; user requirements are usually imprecise or approximate and the application has to interpret them in the right way, in order to retrieve a set of relevant products. In many cases such products are stored as a database of components and an automatic configuration process has to be provided. We propose an approach to intelligent retrieval and configuration of component-based products, starting from a set of possibly fuzzy user requirements provided at different levels of detail. A conceptual product model is introduced and its use during the configuration process is discussed. The proposed approach exploits fuzzy SQL and a configuration process (from basic to complex components) that can be implemented on top of a standard RDBMS. The approach is illustrated through a simple example based on a PC assembly task.

1 Introduction

The wide use of e-commerce applications has produced a remarkable interest in approaches able to provide flexible retrieval of specified items, in such a way that the final user is able to deal with a limited but very significant amount of information. When searching an on-line electronic catalog for a desired product, the user is asked to specify a set of requirements the product has to fulfill [11]; however, very often either the user overspecifies the product (and then no item is returned) or he/she underspecifies the product (and then a very large amount of items is returned). Moreover, several products having a reasonable market on the web are configurable products (e.g. personal computers, travels, music compilations, personal gifts, etc...). This means that the final desired product is composed by a set of components meeting a given set of (possibly imprecise) requirements. However, a limitation of several web applications to e-commerce is the lack of automatic configuration facilities. Very often the deployed application is not able to provide a reasonable help in configuring the product, leaving the user alone in trying to merge sub-components, in order to fulfill every needed requirement. In addition, due to the boolean nature of standard SQL retrieval, soft (i.e. imprecise or approximate) requirements are difficult to be dealt with; only those components that completely satisfy the search parameters are returned, missing those items that partially match the query parameters, but that may be potentially interesting to the user. This problem is particularly worth in B2C (Business to Customer) applications, where the final user is a customer who is usually not an expert in the product field and is not usually able to precisely specify every needed feature or to correctly combine elementary components when a configurable product is the target. On the other hand,

RDBMS provides efficient data management and query facilities, and real-world applications cannot leave aside the fact that the data of interest are usually stored in a relational database.

Configuration problems are traditionally dealt with in AI either within the constraint satisfaction paradigm [4, 12, 17, 8] or by structured logic approaches [5, 9, 15, 10]. Common to every knowledge-based approach to configuration are the following items ([7]): (1) a set of concepts or domain objects, possibly organized in classes; (2) a set of relations between domain objects, in particular taxonomical and compositional relations; (3) a configuration objective or task, specifying the demand and the constraints a created configuration must fulfill; (4) control knowledge for the configuration process.

As recognized in [7], both declarative models and inference machineries must be defined, in order to adequately accomplish the final task. In other words, operational processable models are needed. In order to achieve such a goal, different methodologies can be adopted like compilation of models [14], the use of the case-based reasoning (CBR) paradigm [6, 16], the automatic decomposition of the problem during state space search [1]. However, all the above mentioned approaches do not directly address the issue of flexibly exploiting, during the configuration process, the presence of the data of interest in standard relational databases, and the corresponding data management and query facilities. Moreover, most of the proposed frameworks (exceptions can be found in [13, 6]) do not take into consideration the approximate nature of the user requirements, that very often cannot be assumed to be totally precise. In the present paper, we define an approach for the configuration of products based on the following characteristics:

- a conceptual model of the configurable products, representing the structural decomposition of the modeled product;
- a set of approximate, i.e. fuzzy, user requirements on the desired target product;
- a fuzzy knowledge base providing the semantics for the possible user requirements;
- a fuzzy extension to the standard SQL query language, through which to implement the retrieval and the composition strategy of the product components, directly on top of a RDBMS.

We introduce an extended version of SQL, able to deal with fuzzy predicates and conditions, defined over standard attributes of a table, and based on the **SQLf** language proposed in [2]. We will show how to exploit the conceptual model and the user requirements and constraints, in order to automatically derive a set of fuzzy SQL queries able to retrieve the suitable components. A configuration strategy on the conceptual model is then introduced with the aim of guiding the application to suitably combine the results of such queries, within a standard RDBMS framework.

¹ Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria (Italy), email: luigi.portinale@di.unipmn.it

The paper is organized as follows: section 2 discusses the conceptual model used for the configuration process, section 3 introduces a short review of fuzzy queries on a database, while in section 4 the configuration algorithm is presented and discussed through a specific example. Finally, conclusions are drawn in section 5.

2 The Conceptual Model

As already mentioned, any knowledge-based approach to product configuration needs to define a suitable conceptual model, where both taxonomic (*is_a*) and partonomic (*composed_by*) relations are of fundamental importance. The model we adopt in our approach is inspired by the FPC model by Magro and Torasso [9]. The conceptual framework is based on a hierarchy of components with the following basic ontology: *entities* divided into *categories* representing generic components (i.e. generic classes of components) of the final product, and *basic categories* representing classes of components having specific instances associated with them; *composed_by* links connecting a category *A* to an entity *B*, meaning that *B* is a component of *A*; *is_a* links connecting a category *A* to an entity *B*, meaning that *B* is a subclass of *A* (i.e. that the set of components in *B* are a particular typology of the components in *A*). If *B* is a component of *A* or if *B* is a sub-class of *A* we say that *B* is a child of *A* (i.e. *composed_by* links are from parent to child, while *is_a* link are from child to parent). We also consider cardinality information with respect to a *composed_by* link: in particular every such a kind of link is provided with a so-called *number restriction* ([3]) representing the minimal and maximum number of sub-components that may occur in the link.

An example of a conceptual model for a (simplified) personal computer system is reported in figure 1² (similar examples in the FPC framework are reported in [9, 1, 10]). The PC system is composed by exactly one computational system and exactly one storage system; the computational system can be composed by one or two CPUs, one motherboard (MB) and one set of RAM slots; the storage system of interest is composed by one or two hard-disks and by one (optional) CD-ROM; Different typologies of CPU, MB or RAM are possible, as well as different typologies of hard-disks. For example MBs are differentiated into ASUS and GigaByte MBs, which are in turn both categorized into Intel or AMD MBs. The ASUS entity is a category (i.e. there are no specific stored instances of such category), while ASUSAMD is a basic category with specific stored instances associated to such component. As we will see, instances of non-basic categories are built, during the configuration process, from instances of basic categories. Similar consideration hold for the other entities in fig. 1.

Entities can have attributes. Attributes may be specific to a given entity (i.e. non-inheritable) or inherited from source to target in a *composed_by* link chain and from target to source in a *is_a* link chain. Inheritable attributes are then associated only to categories; in case they are defined on a category having sub-components, an *aggregation function* must be defined; the meaning is that the value of the attribute is determined by combining the values of the inherited attributes of its sub-categories (combination provided by the aggregation function specified). For example the attribute PRICE is defined on the category PC and inherited by every sub-component; however the price of the final product (i.e. the assembled PC) is composed by summing the prices of each component, so the aggregation function SUM (+) is defined over PRICE on category PC. The

aggregation function is inherited along a *composed_by* link chain (the inheritance of the function is stopped where a category has not sub-components). Aggregation function can be standard mathematical function (as SUM for PRICE) or any user-defined function with a specific aggregation task (e.g. the function *assoc_list* on attribute COMTYPE of the storage system, building an associative list with the possible communication standards of the storage devices present in the final product). In figure 1 inheritable attributes are indicated in capital letters (possibly with the associated function).

Besides aggregation functions, arbitrary functions can be attached to attributes in order to compute their value; this is necessary in case the attribute is a non-inheritable attribute defined on a (non-basic) category. In the example of figure 1, function *disc_f* is associated with the attribute discount of PC to compute the available discount for the assembled computer (this may be a simple percentage on the attribute PRICE or a more complex procedure taking into account the kind of involved components). In categories being a target of an *is_a* link, a special *discriminant attribute* is introduced, which aim is to distinguish the different typologies the category has (underlined attributes in fig. 1). For example attribute brand in the category CPU determines the brand of the corresponding component (and so identify the typology below). It is worth noting that the introduction of a typology is a modeling choice, often depending on efficiency reasons: for example the introduction of basic categories IntelCPU and AMDCPU (with *is_a* links to CPU) can be justified when a large number of user queries make a categorized on the brand of the CPU; an alternative could be to make CPU a basic category, with a standard attribute brand of type enum('Intel', 'AMD'). In our framework, a fuzzy semantics can be associated to any attribute, by considering its type (see [11]). The definition of suitable fuzzy sets or operators can then be exploited to model a given level of approximation in the user requirements for the target configured product. This means that the final user can specify both crisp (i.e. precise and well-defined) requirements over an attributes (e.g. "I need an Intel CPU and an hard-disk with a capacity greater than 60Gb"), as well as imprecise (i.e. fuzzy) requirement ("I want a PC with a large memory size and with a price of about \$500").

The definition of the fuzzy sets associated with the attributes can be made at different levels of details. Indeed, the fuzzy semantics of a linguistic value depends in general from the entity we are considering; for instance, a linguistic value "cheap" for attribute PRICE of category PC has a different semantics (and so a different fuzzy set definition) than the same value defined on the same (inherited) attribute on category CPU, since the price of a component may have a different order of magnitude than the price of the whole product. This means that, if fuzzy user requirements are allowed on a given attribute of an entity, possible different fuzzy semantics for the linguistic values are allowed for each entity that defines or inherits the attribute, if the entity is part of a *composed_by* chain. Conversely, entities in a *is_a* chain sharing the same attribute (by inheritance), also share the fuzzy semantics for that attribute. For instance, the fuzzy definition of the concept "high capacity" is shared by entities Hard-disk, IBM and Maxtor.

User requirements represent then a set of user constraints on the required configuration that can be finally expressed through a complex possibly fuzzy condition involving both the set of considered entities, as well as the set of attributes. Moreover, as the notion of configuration assumes, the conceptual model is also augmented with a set of *constraints*. We can identify two different kinds of constraints: (1) *model constraints* (MC) representing general constraints imposed on the model and that must be satisfied

² Dashed squares around some entities will be explained in section 4.

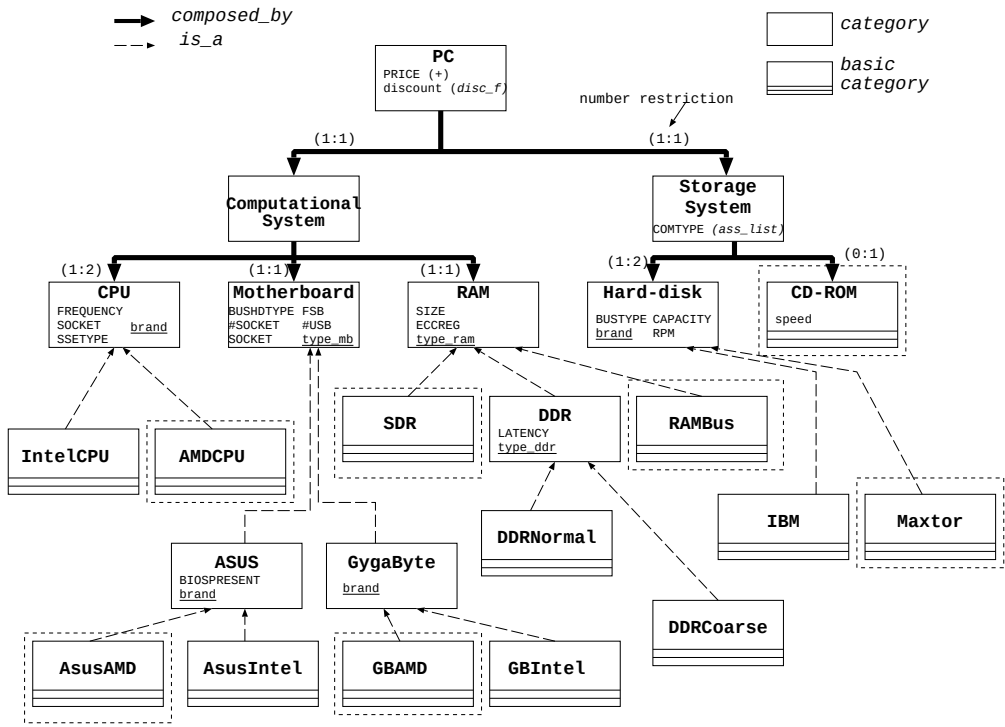


Figure 1. Example of a conceptual model for a PC

by any allowed configurations; (2) *user constraints (UC)* which are specified by the user only for the current configuration process (user's requirements). We assume that a *MC* is an expression according to the following grammar:

```

<mc> ::= (<mc>) | not <mc> | <mc> and <mc> |
<mc> or <mc> | <expr>
<expr> ::= <expr1> <rel> <expr2>
<expr1> ::= <entity>.<attribute> | #<entity>
<expr2> ::= <entity>.<attribute> | #<entity>
| <value>
<rel> ::= <relational.symbol>

```

where <entity> is the name of an entity, <attribute> the name of an attribute, <value> represents any allowed value on the range of an attribute, <relational.symbol> represents any standard relational symbol like equality, diversity, etc..., and #E represents the cardinality (number of occurrences) of entity *E*.

User constraints can be either of the same form of *MCs* or specific condition of selection over component attributes. In the latter they can be either crisp/boolean conditions or fuzzy conditions (see [11] and section 3). Finally, in case a user requires $n > 1$ occurrences of a given entity (as allowed by the corresponding number restriction), then such an entity *E* is supposed to be replicated n times ($E(1), \dots, E(n)$).

Since both model and user constraints can contain fuzzy conditions, we must envision a suitable tool to retrieve items satisfying the specified criteria (see next section).

3 Queries with Fuzzy Conditions

It is well-known that standard relational query languages, like SQL, only support boolean queries. However, in our framework it is nec-

essary to support “fuzzy” or imprecise queries. In particular, we are interested in *fuzzy queries* on an *ordinary (non-fuzzy) database*. We consider, as in [2], the following syntax

```
SELECT ( $\lambda$ ) A FROM R WHERE  $f_c$ 
```

which meaning is that a set of tuples with attribute set *A*, from relation set *R*, satisfying the fuzzy condition f_c with degree $\mu \geq \lambda$ is returned. In fuzzy terms, the λ -cut of the fuzzy relation R_f resulting from the query is returned. The interesting point is that, when the min operator is adopted as *t-norm* for conjunction and the max operator is adopted as *t-conorm* for disjunction, then it is possible to derive from a fuzzy SQL query, an SQL query returning exactly the λ -cut required (see [2] for details).

Example. Consider a generic relation *PRODUCT* containing the attribute *price* over which the linguistic term *medium* is defined. Figure 2 shows a possible fuzzy distribution for *medium* as well as the distribution of a fuzzy operator $\ll$ (much less than), defined over the difference $(a - b)$ of the operands, by considering the expression $a \ll b$. Let the retrieval confidence level be $\lambda = 0.8$; it is trivial

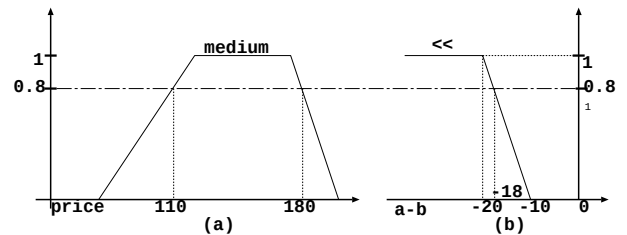


Figure 2. Example of Fuzzy Distributions and λ -cut

to verify that if min is the t-norm for conjunction a condition like $(\text{price} = \text{medium} \wedge \text{price} \leq 100)$ is satisfied at level at least λ iff $(110 \leq \text{price} \leq 180) \wedge (\text{price} - 100) \leq -18$. The latter condition can be easily translated in a standard WHERE clause of SQL.

4 The Configuration Strategy

Given the conceptual model described in section 2, a natural choice is to associate a relation (table) with each basic category; they represent the basic components to be used for the configuration, stored as tuples of the corresponding relation. Data structure corresponding to inner nodes will be created during the configuration process. In the example of figure 1 we will have the following tables (table names represent the path in the model):

```
PC-CS-CPU-INTELCPU(id1,frequency,socket,sstype,price)
PC-CS-CPU-AMDCPU(id2,frequency,socket,sstype,price)
PC-CS-MB-ASUS-ASUSINTEL(id3,biospresent,bushdtype,
#sockets,socket,fsb,#usb,price)
PC-CS-MB-ASUS-ASUSAMD(id4,biospresent,bushdtype,
#sockets,socket,fsb,#usb,price)
PC-CS-MB-GB-GBAMD(id5,bushdtype,#sockets,socket,fsb,
#usb,price)
PC-CS-MB-GB-GBINTEL(id6,bushdtype,#sockets,socket,fsb,
#usb,price)
PC-CS-RAM-SDR(id7,size,eccreg,price)
PC-CS-RAM-RAMBUS(id8,size,eccreg,price)
PC-CS-RAM-DDR-DDRNORMAL(id9,latency,size,eccreg,price)
PC-CS-RAM-DDR-DDRCOARSE(id10,latency,size,eccreg,price)
PC-SS-HD-IBM(id11,bustype,capacity,rpm,comtype,price)
PC-SS-HD-MAXTOR(id12,bustype,capacity,rpm,comtype,price)
PC-SS-HD-CDROM(id13,speed,comtype,price)
```

where fields id_i work as primary keys and are supposed to be available to any entity in the model to identify their components (see below).

Let us suppose to have the following Model Constraints (MC): MC1: "ASUS MBs are incompatible with DDRCOARSE memories"; MC2: "the socket type of CPU and MB must be the same"; MC3: "the number of CPUs must not be greater than the number of sockets in MB"; MC4: "if there are more CPUs they must be the same product"; MC5: "the communication standard of storage devices (HDs and CDROMs) must be the same". Their formalization is reported below:

```
MC1: not(Motherboard.type='Asus' and
DDR.type='Coarse')
MC2: CPU.socket = Motherboard.socket
MC3: #CPU ≤ Motherboard.#sockets
MC4: CPU(1).id_cpu = CPU(2).id_cpu (iff
#CPU=2)
MC5: Hard-disk.comtype = CDROM.comtype
UC1: CPU.brand = Motherboard.brand
```

These are considered as model constraints that must be satisfied by any configuration; We also add a specific user constraint UC1: "CPUs and MB must be of the same brand". Such a constraint can be added to the set of model constraints, but only for the current configuration process.

Once the model and user general constraints are defined, the configuration can start, by collecting specific user requirements as well. We have implemented a configuration algorithm based on the following steps:

1. following initial user requirements, user constraints and model constraints, a part of the general conceptual model is identified and instantiated;
2. starting from basic categories, we ask the user to specify (possibly fuzzy) requirements for the current entity and a fuzzy SQL query is automatically generated;
3. when every child of a given entity has been queried, a view is generated on the parent in the following way: in case of *is_a* links an SQL UNION operation is performed, otherwise an SQL JOIN (checking for constraints) is performed;
4. the procedure is iterated (going to step 2), until a view is generated on the top of the model's hierarchy.

Let us show some details of the above procedure with an example using the model of fig. 1.

4.1 Example: PC configuration

Let us suppose that the initial user requirements are the following: "I'm interested in a single CPU system, in IBM disks, in Intel components and I don't want a CDROM"; The first requirement states that #CPU=1, so no replication is needed and constraint MC4 is not activated. Moreover, the system caches the value #CPU=1, since it will use it later on to check MC3 when needed. The remaining requirements prune from the model all the entities surrounded by dashed squares in fig. 1.

Now the configurator starts from basic categories, asking the user to specify (possibly fuzzy) requirements over categories, by producing suitable (fuzzy) SQL queries. For example the user may require an IBM hard-disk with high capacity with a confidence of 0.7: the following view, representing the possible hard disks the user is looking for is generated:

```
CREATE VIEW PC-SS-HD AS
SELECT (0.7) id11 as id_hd, brand='IBM',
price, comtype
```

FROM PC-SS-HD-IBM WHERE capacity=high where high is a fuzzy linguistic values suitably defined over the attribute of interest (capacity). Notice that only the *id* field (needed to retrieve the components with all their inherited attributes) and the (non-inherited) discriminant attribute are explicitly necessary in such a view. For the sake of convenience, we also store in the view inherited attributes associated to an aggregation function; this avoids the need of a further join condition when *composed_by* links are dealt with (see below). The configuration process switches now to basic categories for RAM and the user is asked to specify requirements over the RAM³; suppose the user decides that the price of the RAM has to be not very expensive with a confidence of 0.7, the following query (building category DDR) is generated:

```
CREATE VIEW PC-CS-RAM-DDR AS
(SELECT (0.7) id9 as id_ddr,
type_ddr='ddrnormal',
price FROM DDRTNormal WHERE
price<>very.expensive) UNION
(SELECT (0.7) id10 as id_ddr,
type_ddr='ddrcoarse',
price FROM DDRCoarse WHERE
price<>very.expensive)
```

Next step generates the RAM view:

```
CREATE VIEW PC-CS-RAM AS
```

³ It is worth remembering that inherited attributes keep their semantics along an *is_a* chain).

```

SELECT id_dds as id_ram, type_ram='ddr',
price
FROM PC-CS-RAM-DDR

```

Concerning the CPU, suppose the user wants a CPU with high speed of the front serial bus (confidence 0.9); the generated query is:

```

CREATE VIEW PC-CS-CPU AS
SELECT (0.9) id1 as id_cpu, brand='Intel',
price FROM PC-CS-CPU-INTELCPU
WHERE fsb=high

```

For the MB, the user asks (confidence 0.9) a high speed front serial bus and a large number of USB ports; these are the generated views:

```

CREATE VIEW PC-CS-MB-ASUS AS
SELECT (0.9) id3 as id_asus, brand='Intel' ,
price FROM PC-CS-MB-ASUS-ASUSINTEL
WHERE fsb=high AND #usb=large

CREATE VIEW PC-CS-MB-GB AS
SELECT (0.9) id6 as id_gb, brand='Intel',
price FROM PC-CS-MB-GB-GBINTEL
WHERE fsb=high AND #usb=large

CREATE VIEW PC-CS-MB AS
(SELECT id_asus as id_mb, type_mb='Asus',
price FROM PC-CS-MB-ASUS) UNION
(SELECT id_gb as id_mb, type_mb='gb', price
FROM PC-CS-MB-GB)

```

At this stage, the configuration algorithm has to deal with compositional links. Concerning the storage system a simple SELECT is sufficient⁴:

```

CREATE VIEW PC-SS AS
SELECT id_hd, price, comtype FROM PC-SS-HD

```

Regarding the computational system a JOIN merging the child components and checking for the available constraints is necessary as follows:

```

CREATE VIEW PC-CS AS
SELECT id_cpu, id_mb, id_ram,
(c.price+m.price+r.price) as price
FROM PC-CS-CPU c, PC-CS-MB m, PC-CS-RAM r
WHERE MC1 AND MC2 AND MC3 AND UC1

```

Constraints can be implemented as follows⁵:

```

MC1: NOT(m.type_mb='Asus' AND r.id_ram IN
(SELECT id_dds FROM PC-CS-RAM-DDR WHERE
type_dds='ddrcoarse'))
MC2: c.socket=m.socket
MC3: m.#sockets >= 1
UC1: c.brand=m.brand

```

Finally, the last view representing the possible PC configurations is generated, by possibly imposing further requirements as, for instance, a medium final price with confidence 0.9:

```

CREATE VIEW PC AS
SELECT (0.9) id_cpu, id_mb, id_ram, id_hd,
(cs.price+ss.price) as price, disc_f as
discount,
(price-discount) as disc_price

```

```

FROM PC-CS cs, PC-SS ss
WHERE disc_price = medium

```

The framework and the configuration algorithm we have described have been implemented in a J2EE 3-tier architecture with JDBC interface. Fig. 3 shows a screenshot of a prototypical web application we developed using the approach described in the present work. Concerning the configuration example discussed above, we have tested it on an Oracle XE database; table 1 reports the cardinalities of the tables in the DB (i.e. the number of stored instances of basic categories).

Table	Cardinality
PC-CS-CPU-INTELCPU	15
PC-CS-CPU-AMDCPU	10
PC-CS-MB-ASUS-ASUSAMD	3
PC-CS-MB-ASUS-ASUSINTEL	3
PC-CS-MB-GB-GBINTEL	5
PC-CS-MB-GB-GBAMD	3
PC-CS-RAM-DDR-DDRNORMAL	7
PC-CS-RAM-DDR-DDRCOARSE	4
PC-CS-RAM-SDR	3
PC-CS-RAM-RAMBUS	3
PC-SS-CDROM	4
PC-SS-HD-IBM	4
PC-SS-HD-MAXTOR	4

Table 1. Number of basic components (occurrences of basic categories) in the PC assembly example application.

Given the above number of instances, the PC configuration task discussed in this section, would produce, without considering constraints, **66640** possible configurations (i.e. $1\text{CPU}+1\text{MB}+1\text{RAM}+1\text{HD}=35 \cdot 14 \cdot 17 \cdot 8 = 66640$). By considering the given model and user constraints, the number of possible solutions reduces to **4560**.

User requirements collected during the configuration process further reduces such a number. In particular, table 2 reports the cardinalities of the generated views. We can notice that user requirements,

View	Requirement (conf. lev.)	Cardinality
PC-SS-HD	capacity=high (0.7)	1
PC-CS-RAM-DDR	price<>very_expensive (0.7)	3
PC-CS-RAM	none	3
PC-CS-CPU	fsb=high (0.9)	11
PC-CS-MB-ASUS	fsb=high^#usb=large (0.9)	5
PC-CS-MB-GB	fsb=high^#usb=large (0.9)	3
PC-CS-MB	none	8
PC-SS	none	1
PC-CS	none	88
PC	price=medium (0.9)	5

Table 2. Cardinalities of the views generated during configuration for the PC assembly example application.

even if approximate, can reduce in a significant way the number of possibility to be considered. Of course, the confidence level used for a given fuzzy requirement is also relevant for selecting a reasonable number of results. In particular, in the above example, when building the view concerning to the computational system (CS), the corresponding join operation produces 88 tuples (they results from 3 RAMs modules, 11 CPUs and 8 MBs, pruned out by the available constraints); by requiring a medium price for the final PC (with a

⁴ Notice that if price and comtype would not been stored in the PC-SS-HD view, a join on id attributes with PC-SS-HD-IBM would have been necessary to retrieve such aggregated attributes.

⁵ Notice that if the user had chosen #CPU=2, then constraint MC4 would have also been added here. In addition, constraint MC3 is, in this example, trivially true (any motherboard has at least 1 socket), while user constraint UC1 could be safely removed, since from the initial user requirements (user's exclusive interest in Intel devices) it is definitely satisfied.

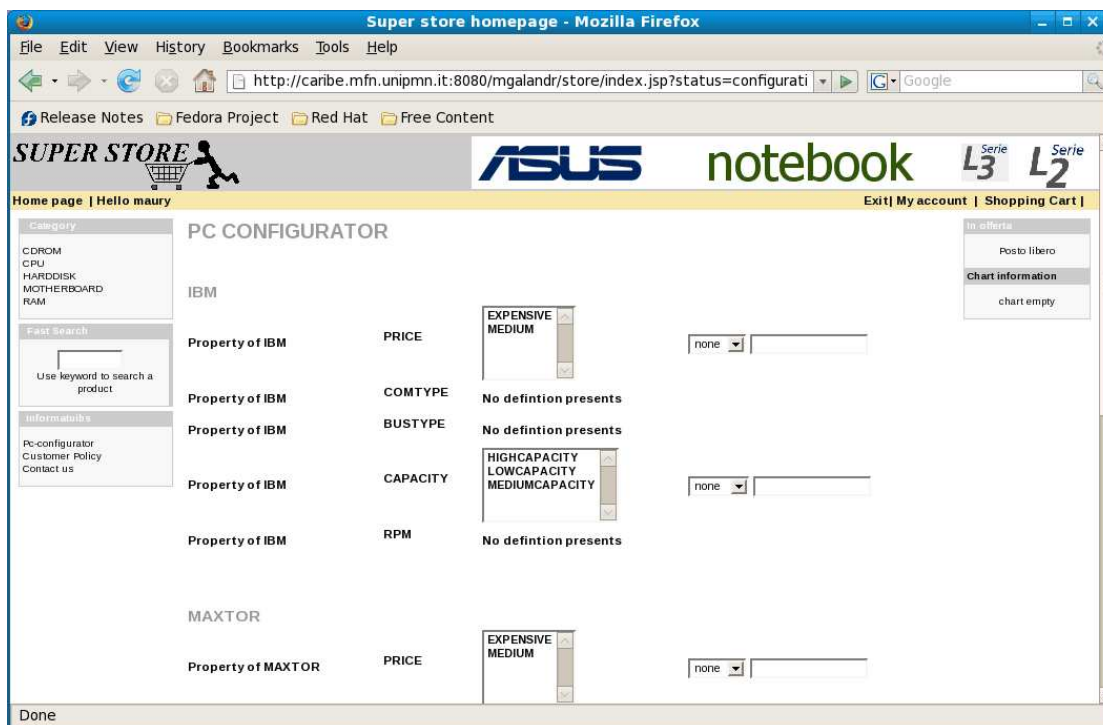


Figure 3. A screenshot for the PC assembly web application.

confidence of 0.9), only 5 tuples are finally presented to the user. It is worth noting that it would be really important to have the possibility of measuring the sensitivity of the confidence threshold with respect to the number of returned tuples, in such a way of having the possibility of defining suitable, possibly interactive, configuration policies able to tune such a threshold in a flexible way.

5 Conclusions and Future Works

We have presented an approach to product configuration, based on a hierarchical conceptual model and on fuzzy user requirements on the product features. The approach can be implemented on top of a relational database, exploiting the whole power of an SQL engine to implement both retrieval of product components, component composition and constraint checking. The proposed framework is a first step towards the definition of a flexible configuration architecture, where suitable strategies of system-user interactions can be defined. Indeed, future works will concentrate on such strategies, in such a way of providing the user with the guarantee of getting a reasonable set of acceptable configurations, by checking when constraint are too strict or too large, by defining suitable policies on the definition of either hard or soft requirements, by tuning the sensitivity of the fuzzy semantics of attributes and by allowing a suitable ranking (possibly exploiting fuzzy membership) of the obtained solutions.

REFERENCES

- [1] L. Anselma, D. Magro, and P. Torasso, 'Automatically decomposing configuration problems', in *LNAI 2829*, 39–52, Springer, (2003).
- [2] P. Bosc and O. Pivert, 'SQLf: a relational database language for fuzzy querying', *IEEE Transactions on Fuzzy Systems*, 3(1), (1995).
- [3] R. Brachman and J. Schmolze, 'An overview of the KL-ONE knowledge representation system', *Cognitive Science*, 9(2), 171–216, (1985).
- [4] G. Fleischanderl, G. Friedrich, A. Haselbock, H. Scheiner, and M. Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, 13(4), 59–68, (1998).
- [5] G. Friedrich and M. Stumptner, 'Consistency-based configuration', in *Proc. AAAI Workshop on configuration*. AAAI Press, (1999).
- [6] L. Geneste and M. Ruet, 'Fuzzy case-based configuration', in *Proc. ECAI'02 Workshop on Configuration*, pp. 1–10, Lyon, FR, (2002).
- [7] T. Krebs, L. Hotz, and A. Gunter, 'Knowledge-based configuration for configuring combined hardware/software systems', in *Proc. PUK 2002*, Freiburg, GE, (2002).
- [8] D. Magro, 'Using constraint optimization to enhance the diversity in the set of computed configurations', in *Proc. ECAI'06 Workshop on Configuration*, Riva del Garda, IT, (2006).
- [9] D. Magro and P. Torasso, 'Supporting product configuration in a virtual store', in *LNAI 2175*, 176–188, Springer, (2001).
- [10] D. Magro and P. Torasso, 'Decomposition strategies for configuration problems', *AIEDAM Journal*, 17(1), 51–73, (2003).
- [11] L. Portinale and S. Montani, 'A fuzzy case retrieval approach based on SQL for implementing electronic catalogs', in *LNAI 2416*, 321–335, Springer, (2002).
- [12] D. Sabin and E. Freuder, 'Configuration as composite constraint satisfaction', in *Proc. Artificial Intelligence and Manufacturing Research Planning Workshop*, pp. 153–161, Albuquerque, NM, (1996).
- [13] R. Schaefer, 'Rules for using multi-attribute utility theory for estimating user's interests', in *Proc. ABIS'01 Workshop*, (2001).
- [14] C. Sinz, 'Knowledge compilation for product configuration', in *Proc. ECAI'02 Workshop on Configuration*, pp. 23–26, Lyon, FR, (2002).
- [15] T. Soininen, I. Niemela, J. Tiihonen, and R. Sulonen, 'Representing configuration knowledge with weight constraint rules', in *Proc. AAAI Spring Symposium on Answer Set Programming*. AAAI Press, (2001).
- [16] H-E. Tseng, C-C. Chang, and S-H. Chang, 'Applying case-based reasoning for product configuration in mass customization environments', *Expert Systems with Applications*, 29(4), 913–925, (2005).
- [17] M. Veron and M. Aldanondo, 'Yet another approach to CCSP for configuration problem', in *Proc. ECAI'00 Work. on Configuration*, pp. 59–62, Berlin, GE, (2000).

Constraint-based Modeling and Exploitation of a Vehicle Range at Renault's: Requirement analysis and complexity study

Jean Marc Astesana¹, Yves Bossu¹, Laurent Cosserat¹ and H       Fargier²

Abstract. As outlined by [10] "constraint programming is well-suited to treat complex configuration constraints such as compatibility and requirement- constraints in addition to standard arithmetic, comparison, and boolean constraints". As a consequence several configurators have been implemented that are based on this technology. Nevertheless, configuration applications are not restricted to the sole problem of on-line product configuration. Different business tasks may occur at different steps of the life cycle of the product range, e.g. the design of the model, the management of the Bill of materials, etc. In the present paper, we analyse and formalize a series of the business needs that arise around the problem of car configuration at Renault's, the French car manufacturer. It appears that these tasks call for a series of requests to a constraint satisfaction problem (CSP) that often differ from the ones classically handled by the constraint programming frameworks. This application actually appears as a fruitful source of research problematics for both the CSP community and the Configuration community. We aim at motivating this research, by a formalization and a theoretical study, in terms on complexity, of the series of requests risen by this application.

1 Introduction

The product range, or "product diversity", of the vehicle diversity is very broad, with a huge number of different vehicles for each single model: for instance, there are about 10^{21} possible "Trafic" vehicles (small delivery vans). It is of course not possible to list all these vehicles in a database ; this set of vehicles has therefore to be represented in intention. The possible vehicles in product range of a given model, e.g. the Trafic model, are defined by finite domain variables, for instance the variable *fuel.type*, whose values are $\{petrol, diesel, LPG\}$. The values available for a variable generally depend on the ones assigned to other variables: the possible options, for instance, typically depend on the entry-level versions of the vehicle. Such dependencies are specified by constraints between the variables. In other terms, the diversity of the model is defined by a Constraint Satisfaction Problem (CSP). Different business tasks occur at different steps of the life cycle of the product range associated to a given model: the modeling and the documentation of the vehicle range, the management of the Bill of materials, the on-line exploitation of the model by end customers, i.e. the task of configuration.

As outlined by [10] "constraint programming is well-suited to treat complex configuration constraints such as compatibility and

requirement- constraints in addition to standard arithmetic, comparison, and boolean constraints". As a consequence several configurators have been implemented that are based on this technology [1, 19, 10], taking advantage of the search algorithms and local consistency techniques such as arc-consistency provided by these formalisms. The latter type of algorithm suits interactive applications since it reduce the domains from which users pick their choices. (Preference) - based search algorithms allow to get a complete configuration from a specification of the user's preference. The CSP toolbox has been extended with conflict search algorithms in order to provide the user with explanations of the inconsistency of her choices [12].

Nevertheless, configuration applications should not be restricted to the sole problem of on-line product configuration. The analysis of the business needs that arise around the problem of car configuration and more generally the management of the vehicle range at Renault lead to the definition of other tasks. All these tasks call for a series of requests to the CSP, that often differ from the ones classically handled by the constraint programming frameworks.

This application thus appears as a fruitful source of research problematics. In the present paper, we aim at motivating this research, by formalization and a theoretical study, in terms on complexity, of the series of requests risen up by this application. Section 2 presents the specificities of the constraint based modeling of a vehicle product range. Section 3 is a general description of the application. Section 4 then deals with its formalization in terms of CSP-oriented requests and presents our complexity study.

2 Representing the vehicle range as a CSP

2.1 Notations and background

A CSP is classically defined as triplet $P = \langle X, D, C \rangle$. For any variable x in X , $D(x)$ is the (finite) domain of x . For any constraint c in C , $vars(c)$ denotes the set of variable involved in c .

A *assignment* d is a function that maps any $x \in X$ to a value in its domain or to the symbol $*$ ($d(x) = *$ means that the variable has not yet been assigned a value). d is complete iff it does not involve this symbol. $vars(d)$ is the set of the variables assigned by d (i.e. the x such that $d(x) \neq *$) We say that d is a complete assignment of $Y \subseteq X$ iff for any $x \in Y$, $d(x) \neq *$. $D(Y) = \{d, \forall x, d(x) = * \leftrightarrow x \notin Y\}$ is the set of complete assignments of Y . Finally, we say that d' extends d (to $vars(d')$) iff $\forall x \in vars(d)$, $d(x) = d'(x)$ - we write this $d' \models d$.

A constraint c involves a set $vars(c)$ of variables and can be viewed as a function c from the set of assignments of $vars(c)$ to

¹ Renault SA, 13 avenue Paul Langevin 92359 Plessis Robinson email: jean-marc.astesana@renault.com

² IRIT/RPDMP, France, email:fargier@irit.fr

$\{\top, \perp\}$: $c(d) = \top$ iff d satisfies the constraint. No assumption is done about the way used to represent the constraints: it is only assumed that, for any d assigning (at least) all the variables of c , it can be checked in polytime, ideally in linear time, whether d satisfies c (denoted $d \models c$) or violates it ($d \not\models c$).

A solution of P is a complete assignment of X that satisfies all its constraints. $Sols(P)$ is the set of its solutions. P is *consistent* iff $Sols(P) \neq \emptyset$, otherwise, it is *inconsistent*. $Sols(P)^{\downarrow Y} = \{d \in D(Y) \text{ s.t. } \exists d' \in Sols(P), d' \models d\}$ denotes the *projection* of $Sols(P)$ on $Y \subseteq X$. The CSP is globally consistent iff its domains are globally consistent, i.e. iff $\forall x \in X, D(x) = Sols(P)^{\downarrow \{x\}}$. Finally, we say that a additional constraint c (not necessarily in P) is *valid* (or inferred by the CSP) iff $\forall d \in Sols(P), d \models c$.

Some constraints will be represented as boolean expressions on fluents - this is typically the case of the option constraints that relate the versions of the vehicle to the option constraints (c.f. Section 2.2). A *fluent* is a couple (x, A) , where $x \in X$ and generally, $A \subseteq D(x)$, but this is not compulsory; the fluent is *elementary* when A is a singleton - it then represents an assignment of x . A *boolean expression* or *boolean condition* is a formula built on fluents using the logical operators $\vee, \wedge, \implies, \neg, \leftrightarrow$. $vars(c)$ is the set of variables used by the boolean condition c . An assignement d such that $x \in vars(d)$ satisfies the fluent $f = (x, A)$ iff $d(x) \in A$ (this is denoted $f(d) = \top$). For any assignment d of $vars(c)$, the values it returns for d , denoted $c(d)$ is computed according to the usual semantics of the logical operators.

As previously said, many of the constraints used in the specification of the range will be expressed as boolean conditions. Such expressions will also be used in other tasks and requests, typically for the definition of the Bill of materials (c.f. Section 3.3). Since it is the case in our application, we assume each of the boolean expressions handled in the paper is internally consistent ($\exists d \in D(vars(c)), c(d) = \top$) and that the satisfaction (or violation) of an expression by an assignement can be checked in polytime.

2.2 Specificities of the constraint-based representation of a vehicle range

Several extensions of the CSP paradigm have been proposed in order to handle the constraint-based definition of a catalog or a range of (configurable) products. These extensions have been motivated by difficulties and characteristics that are specific to the modeling and the handling of catalogs of configurable products. Dynamic CSPs [13], for instance suit the problems where the existence of some optional variables depends on the value of another variable. Other extensions include composite CSPs [15], interactive CSPs [9], hypothesis CSP [2], generative constraint satisfaction [18, 8], etc. Some of the characteristics handled by these formalisms may happen in our application, but they are marginal (for instance, almost all the variables must be assigned a value, so no distinction is to be made between "mandatory" and "optional" variables).

The representation of the data is thus not the point of the paper. In the following, assume that the product range is specified by a classical CSP, and we focus on the requests that are addressed to it.

As it is the case with many real-life instances, the (again, classical) CSPs considered in our application have special features, both at the semantic level (type of variables, type of constraints) and syntactical level (characteristics of the constraints graph). Renault's range indeed follows the so-called version-option modeling. According to this modeling, the CSP variables are hierarchized in two sets:

- The *major variables* are used to define versions. They form a subset $Maj \subset X$. $Sols(P)^{\downarrow Maj}$, the restriction of $Sols(P)$ to the variables of Maj , defines the set of versions of the range. A particular variable sometimes encodes the version itself, but it may happen that the version exists only in an implicit way, as a combination of major variables such as engine or gearbox, for instance.
- All other variables define features that depend on the version. Let us call them *minor variables*. They form a distinct subset Min of X . A given value in the domain of such a minor variable may be unavailable on a given version, possible on a second other one and mandatory on a third one. This is typically the case of the value *sunroof* in the domain of the minor variable *roof_type*.

Several different types of constraints are involved:

- The *major constraints* restrict the values of major variables. A major constraint is generally defined by a table, i.e. by the explicit list of the valid combinations of values for its variables.
- The *"option" constraints* link minor variables to major variables. They are of the form " $(x = v) \implies (y \in A)$ ", where x is a major variable, v is a value of x , y is a minor variable. For instance, suppose that specific radios are proposed as options but only in top-of-the-range versions: an option constraint specifies the radios that are available for each version.
- The *Pack constraints* are used to gather several options into a single one - this composite option is called a pack and the original options are called the components of the pack. The simplest example is the constraint that models the fact that selecting a pack means selecting all its components: $(x = pack_j) \implies \bigwedge_{i=1}^n (x_i = opt_i)$.
- *Miscellaneous constraints* gather constraints that do not belong to any of the previous categories. In practice, these constraints involve 2 to 4 major variables (in addition to their minor variables).

The way in which the constraints are represented is not restricted. Constraints may be in particular expressed as boolean expressions, e.g. the constraint $(fuel_type = diesel) \wedge (gearbox = automatic) \implies (heating = air_conditioning)$. It is also possible to use a purely CNF oriented description of the range, where all the variables are boolean but the present paper does not impose this restriction.

The complexity of the CSP is mainly impacted by:

- The number of major and minor variables. There are generally about 10 major variables, and no more than 150 minor variables.
- The number of different valid combinations of major variables: this number is generally lower than 100 when for the commercial ranges, but it can exceed 10,000 for some technical ranges.
- The existence of packs. The definition of packs indeed leads to many constraints, that may dramatically increase the complexity of the CSP. As a matter of fact, consider that a given option may be involved as a component of several packs, these packs being mutually exclusive. Packs are at worst up to 10. Each of them involves most often between 2 and 5 components.
- The existence of miscellaneous constraints - they may be up to 100. Unlike major and option constraints, that link variables in a hierarchical way and define an (hyper) tree, these constraints can relate any variables, increasing the complexity of the graph.

3 Design and exploitation of the vehicle range

We now describe the business requests that occur at the different steps of the lifecycle of a vehicle range : design, debug and use.

3.1 Modeling the vehicle range

The first task is to properly achieve the modeling of the range by a set of variables and constraints. The model must be consistent in a broad sense: many constraints have actually a double meaning. Following the standard semantics of constraints, the first one is negative: constraints forbid combinations. The second one is positive: the possibilities left by some constraints have to be effective. Let us assume, for instance, that a constraint means "The heating type of vehicles with type M3 engine may be manual or automatic". If some other constraint excludes the vehicles with manual heating, the specification of the range is considered as inconsistent ; we call this property *positive consistency*. The main requests of the modeling task are:

- *(Contextual) consistency of the range*: is there at least one vehicle that satisfies the constraint-based specification ? Is there at least one vehicle that is consistent with a given partial assignment of the variables? More generally, is there at least one vehicle consistent with a given boolean expression upon elementary expressions ($x = value$) (e.g. the expression "heating = manual") ?
- *Conflicts*: When the range is not consistent, the inconsistency has to be explained, which means finding a set of inconsistent constraints, that is minimal w.r.t. inclusion.
- *Contextual Conflicts*: Given a given partial assignment, find a minimal set of constraints inconsistent with this assignment.
- *Positive consistency of the constraints*: For any constraint c , and any tuple in $D(vars(c))$ allowed by the constraint, is there a vehicle in the range that implements this possibility ? Similarly, for any variable and any value in its domain, is there a vehicle in the range that assigns this value to the variable ?

At the end of the step of modeling of the range, the CSP is globally consistent, and obviously, consistent. All the requests that are described in the next sections deal with a (globally) consistent CSP.

3.2 Reporting the vehicle diversity

Once the range has been consistently defined, the Renault experts look for the extraction of relevant and summarized data. The idea is to select some variables (generally, less than 6) and ask the software for a summarized description of the list of all the valid combinations of values of these variables; the summarized descriptions are typically boolean expressions on simple atoms of the form ($variable = value$) or ($variable \in subdomain$), i.e on fluents. The main requests involved by the task of report of the vehicle diversity are:

- *Validity of a summarized description*: Given a summarized description, does all the vehicles of the range comply with it ?
- *Simplification of a summarized description*: Is a summarized description minimal, i.e. is it possible to remove a variable from it without altering its validity ?
- *Generation of summarized descriptions*: Given a set of variables, what are the combinations of their values that correspond to at least one vehicle in the diversity ? the task is here to compute a (minimal) projection of the range of the variable.
- *Counting*: How many vehicles does the range contain ? How many vehicles consistent with a given assignment (or more generally with a boolean expression) ? How many combinations of values for a given subset of variables (e.g. how many different vehicles does the range contain, this number being taken independently of the commercial name, color and country variables)?

3.3 Management of the Bill of materials

The definition of the Bill of materials (BOM) is not included in the specification of the product range. This information, that specifies on which vehicles the different parts are set up, is crucial from a business point of view. The next task deals with the design and debug of the BOM. This task takes place after the specification of the range (Section 3.1) : at this step of the application the CSP is thus globally consistent. The management of the BOM then adds new variables, and new constraints, to this CSP.

The Bill of materials is indeed organized in "generic parts": a generic part is a function fulfilled by a part. For instance, radio, as a function, is a generic part which may be fulfilled by different radios, as parts. The relationships between the CSP-based specification of the range, the generic parts and the corresponding, real parts, is defined by boolean expressions called "use cases": the use case of a part p_i is a boolean expression (over the variables of the vehicle range) specifying on which vehicles the part is set up. Generic parts can be regarded as extra variables, whose values are associated parts $p_1, \dots, p_n$, including sometimes a special one that encodes the absence of the function (if the vehicles do not necessarily have a radio, the absence of radio is considered as a part). Other modelings are possible (e.g. resorting to description logic [?]), but the Renault problem is simple enough to be captured by the classical CSP model without any increase of complexity; in particular, the physical parts $p_1, \dots, p_n$ are not decomposed further.

For a given generic part, the Bill of materials is consistent iff each vehicle implements exactly one part (i.e. satisfies exactly one use case), and if each part is used in at least one vehicle. The following requests occur during the design and debug of the BOM:

- *Concision of the BOM*: For any given part, is there at least one vehicle in the range that uses it ?
- *Exhaustivity of the BOM*: Is any vehicle in the range using a one the parts of a given generic part ? Some vehicles are violating this condition generate a boolean condition representing them.
- *Exclusivity*: Do some vehicles of the range simultaneously use two (or more) of the parts of a given generic part ? If yes, generate boolean condition representing this set of vehicles.

The expressions produced to explain the non-exhaustivity or the non-exclusivity of the BOM should involve as few variables as possible.

3.4 On-line configuration

The topic of constraint-based configuration has been widely dealt with in the litterature (see [13] [18] [15] [8][17] [2] [11], among others). The key idea in on-line configuration is that a customer, or a business-oriented user, step by step defines her product by choosing interactively values for the variables. She can also backtrack by relaxing some of her choices. After each action of the user, the system updates the domains so as to rule out the values that are inconsistent with the current choices (e.g. by listing them in grey rather than black). However, the user may select a ruled-out value, thus making the CSP inconsistent. Roughly, the requests involved by the task of on-line configuration are:

- *Incremental global consistency*: Ensure, after each modification of the choices that the domains are globally consistent.
- *Conflict analysis*: In case of inconsistency, identify a (resp. all the) (minimal) subsets of choices that are inconsistent, or inconsistent with a given partial assignment of the variables.

- *Assessment of the price, assessment of the delay:* Compute the minimum price (resp. delay) of the vehicles of the range that are consistent with the current choices.
- *Completion of the configuration:* As soon as some variables have been set, the user can ask for a completion of the configuration. The system must then search for a vehicle consistent with the current choices (and upon demand, optimizing price or delay).

The requests dealing with the price and the delay deserve some details. The price of a vehicle is actually a sum of elementary costs, each of them being associated with a value of a variable or a boolean expression (the elementary cost can depend on several variables). Notice that an elementary cost may be negative, because it is possible to configure a vehicle with less options than the standard one. The delay is also a sum of elementary delays: the industrial delay (which specifies when the assembly will actually begin), the assembly delay, the transportation delay, etc. The transportation delay, and actually each of the other delay typically depend on the localization of the plant that will assemble the vehicle. This parameter is not a variable of configuration. In the present presentation, we simplify the description, assuming that this localization has been chosen upstream. All the delays but the industrial one can then be considered as constants. Concerning the industrial delay, it depend on the time windows in which the different options will be available. To this extent, a distinguished variable x_t is added to the CSP. The information relative to the delay is encoded by boolean expressions of the form: $c_i \implies x_t \in UI_i$, IU_i being a union of intervals and c_i a boolean expression (eg: $color = green \implies x_t \in (-\infty, 8] \cup [10, +\infty)$).

3.5 CPU time

With respect to interactive applications, e.g. on-line configuration, the CPU time must be (at worst) about one second. Office off-line applications can take hours; but if they have to process many requests, the CPU time assigned to each request is about one second or even one millisecond. However it is possible to take a longer time for especially difficult requests (say, minutes) as long as the global time is not appreciably increased. The requests related to the design of the vehicle range, as well as the one related to the definition of the BOM are proceeded within this order of magnitude.

4 Formal modeling of the request and complexity

The present Section presents a formalization and a complexity analysis of the requests detailed in Sections 3.1 to 3.4. Some of these requests clearly define decision problems in the sense of complexity theory, e.g. problem of consistency of the product range: it obviously comes down to CSP satisfiability. Others are NP-difficult search problems. The complexity of such problems directly depends on (1) the complexity of the task of deciding the existence of the object(s) searched for (e.g. a solution of the CSP, a minimal inconsistent set of constraints) and (2) the complexity of checking the correctness of the object. In some cases, the check is tractable but the problem of existence is intractable (e.g. when searching for an assignment satisfying all the constraints); in some others, deciding of existence of the object searched for is easy, while checking its correctness is a difficult task (given an inconsistent CSP, checking that a subset of constraint is minimal inconsistent is intractable, but deciding the existence of such a subset is trivial).

4.1 Designing the vehicle range

Since the range is modeled by a CSP, deciding whether the range is consistent is a classical problem of satisfiability of a CSP, hence its NP-completeness. In the contextual variant the CSP is consistent, as well as the boolean expression (say, c) which represents the vehicles the user is interested in. However, deciding whether the range contains vehicles satisfying this requirement is a NP-difficult problem:

Proposition 1 ³ *Given a consistent CSP $P = \langle X, D, C \rangle$ and a consistent boolean condition c , deciding whether $\langle X, D, C \cup \{c\} \rangle$ is consistent is a NP-complete problem. The results still hold when c is a fluent and P is consistent.*

Unsurprisingly, a *conflict* is basically a set of fluents that is inconsistent with a set of constraints, and conflicts that are minimal w.r.t. inclusion are searched for. One recognize the notion on minimal unsatisfiable subset used in AI since the late 80's, e.g. [5][6] and more recently in configuration [7][2] (for contextual conflicts, see in particular [6] and [7])

Definition 2 *Given a CSP $\langle X, D, C \rangle$, a minimal conflict is a subset $C' \subseteq C$ s.t. $\langle X, D, C' \rangle$ is inconsistent and $\forall C'' \subsetneq C'$, $\langle X, D, C'' \rangle$ is consistent.*

Given a consistent CSP $\langle X, D, C \rangle$ and a set of constraints $Context$ such that $\langle X, D, C \cup Context \rangle$ is inconsistent, a minimal contextual conflict is a subset $C' \subseteq C$ such that $\langle X, D, C' \cup Context \rangle$ is inconsistent and $\forall C'' \subsetneq C'$, $\langle X, D, C'' \cup Context \rangle$ is consistent.

Finding a conflict is a NP-hard problem. Testing whether a subset of clauses (or constraints) is minimal inconsistent is a well-known BH_2 problem. Roughly, the complexity class BH_2 is defined by $BH_2 = \{L \in L_1 \cap L_2 \mid L_1 \in NP, L_2 \in CO - NP\}$. The hardest problems of this class are referred to as BH_2 -complete problems. Among them is the SAT-UNSAT problem that consists in determining, given a pair of CNF whether the first one is satisfiable and the second one unsatisfiable. The result extends to contextual conflicts.

Proposition 3 *Given CSP $P = \langle X, D, C \rangle$, a set of constraints $Context$ such that $P = \langle X, D, C \cup Context \rangle$ is inconsistent, and $C' \subseteq C$, deciding whether C' is a minimal contextual conflict is a BH_2 -complete problem. The result still holds when P is consistent. It also holds when C' is a set of fluents.*

Finally, the notion of positive consistency is clearly a notion of global consistency of a constraint w.r.t. a consistent set of constraints.

Definition 4 *Given a consistent CSP $P = \langle X, D, C \rangle$ and a constraint $c \in C$, c is globally consistent iff, for any $d \in D(\text{var}(c))$, $d \models c$ iff $\exists d' \in \text{Sols}(P), d' \models d$*

Proposition 5 *Given a CSP $P = \langle X, D, C \rangle$ and a constraint $c \in C$, deciding whether c is globally consistent is a NP-hard problem. It is NP-complete whenever the $\text{Card}(\text{vars}(c))$ is bounded. The result still holds when P is consistent and c is a unary constraint.*

In summary, the requests considered in this task are NP-hard, even when considering that the basic CSP (the range) is consistent. Most of them amount at a *sequence* of consistency checks on a sequence of CSPs. Generally, the CSPs in the sequence do not differ from each other but for one constraint.

³ Proofs have been omitted for lack of space but can be found at [ftp://ftp.irit.fr/IRIT/RPDM/PapersFargier/wsecai10.pdf](http://ftp.irit.fr/IRIT/RPDM/PapersFargier/wsecai10.pdf).

4.2 Reporting the vehicle diversity

This task aims at building summarized descriptions of the vehicle range - at inferring valid information - under the form of a boolean combination of fluents. The summarized descriptions used in this task are intrinsically consistent (the consistency of a sole description is not a source of complexity). Let us first consider the question of the test of validity of such a description. Since capturing the classical problem of clausal inference as a particular case, the request is CO-NP complete. The problem is also closely related to the test of redundancy of a constraint, which is known to be CO-NP complete [3]: c is valid iff it is redundant w.r.t. the CSP.

Proposition 6 *Given a consistent CSP and a consistent boolean condition c , deciding whether c is valid is a CO-NP complete request.*

With respect to the question of generation of summarized description, we consider the request of checking whether a description does coincide with the projection of the range, i.e. of $Sols(P)$, on the variables of interest - the question is Π_2^P -complete:

Proposition 7 *Given a consistent CSP $\langle X, D, C \rangle$ and a consistent boolean condition c , deciding whether c is equivalent to $Sols(p)^{\downarrow vars(c)}$ (i.e. whether $\forall d, c(d) = \top \Leftrightarrow d \in Sols(P)^{\downarrow vars(c)}$) is a Π_2^P -complete request.*

This proposition studies the question of recognizing a projection. Beyond this decision problem, projecting the CSP is a function problem that can be addressed by variable elimination algorithms.

Once a descriptor has been built that is valid or better, that is a projection of the diversity on the variables of interest, we would like it to be minimal with respect to the number of variables it involves.

Definition 8 c is minimal if there is no $x \in vars(c)$ such that the satisfaction of c does not depend on the value given to x , i.e. no $x \in vars(c)$ such that for any $d, d' \in D(vars(c))$ if $d(y) = d'(y) \forall y \neq x$, then $d \models c \Leftrightarrow d' \models c$.

Assuming that the number of variables in $vars(c)$ is bounded, as it is the case in our application, deciding whether c is minimal is a polynomial task. Indeed, the tuples in $d, d' \in D(vars(c))$ can then be listed in polytime. Remark that the request is probably not polytime when $Card(vars(c))$ is not bounded. The last request, the counting of the diversity, is a classical problem of counting the number of solutions of a CSP, known to be $\#P$ complete.

4.3 Management of the Bill of Materials

As said in Section 3.3, the idea is to represent each use case of a generic part by a consistent boolean condition (the internal consistency of the use cases is not a source of complexity);

Definition 9 *Given a CSP $P = \langle X, D, C \rangle$ and C' the specification of a generic part, i.e. a set of (consistent) boolean conditions:*

- C' is concise iff $\forall c' \in C', \exists d \in Sols(P), d \models c'$;
- C' is exhaustive iff $\forall d \in Sols(P), \exists c' \in C' \text{ t.q. } d \models c'$;
- C' is a pairwise exclusive set of use cases iff $\forall c', c'' \in C', \langle X, D, C \cup \{c', c''\} \rangle$ is inconsistent.

I.e. the BOM is concise w.r.t. the generic part considered whenever for any $c' \in C'$, the set of use cases associated to this part, $\langle X, D, C \cup \{c'\} \rangle$ is consistent, and it is exhaustive whenever $\langle X, D, C \cup \bigcup_{c' \in C'} \{c'\} \rangle$ is inconsistent.

Proposition 10 *Given a CSP P and a set of boolean conditions C' ,*

- *The problem of deciding whether C' is concise w.r.t. P is NP-complete.*
- *The problem of deciding whether C' is exhaustive (resp. pairwise exclusive) w.r.t. P is CO-NP complete.*

The result still holds when P is consistent, as it is the case for the product range once the task of design has been achieved.

When the BOM fails to satisfy the exclusivity condition, the user needs to describe the vehicles that make the condition false. The idea is to generate a boolean condition c that capture these vehicles. The task is not that difficult, provided that the pair of non exclusive use cases has been identified : it is enough to make the conjunction of the boolean description of the two use cases. Let c denote this condition. Many of the vehicles satisfying the new condition do not belong to the range. The task is then to consider the CSP $P' = \langle X, D, C \cup \{c\} \rangle$ and to project $Sols(P')$ on $vars(c)$. We have seen in the previous section that this task is intractable (e.g. deciding whether a boolean condition is a marginalization of $Sols(P')$ is a Π_2^P complete problem). The same result holds for the characterization of the vehicles falsifying the exhaustivity condition. Defining a boolean expression c as the disjunction of the negations of the use cases is easy, but many vehicles satisfying c do not belong to the range: projecting $Sols(\langle X, D, C \cup \{c\} \rangle)$ on $vars(c)$ is an intractable task. Given an additional boolean condition c , the problem of deciding whether c is a concise representation of the uncovered vehicles of C' (resp. of non exclusive use cases in C') is thus NP-hard.

4.4 On-line configuration

We have seen that the global consistency of the domains is ensured upstream, before the configuration session, in order to remove from the domains any value that cannot lead to a solution. It must also be ensured during the configuration session, after each (de)assignment of some variable by the user. Hence the notion of global consistency w.r.t. an assignment:

Definition 11 $P = \langle X, D, C \rangle$ is said globally consistent w.r.t. a partial assignment d iff $\forall x \in X, \forall v \in D(x) \exists d' \in Sols(P)$ such that $d' \models d$ and $d'(x) = v$.

Hence P is globally consistent iff it is globally consistent w.r.t. the empty assignment (d , s.t. $d(x) = *, \forall x$). Recovering the global consistency after the assignment of some variables is intractable:

Proposition 12 *Given a CSP $P = \langle X, D, C \rangle$, and a (partial) assignment d , deciding whether P is globally consistent w.r.t. d is a NP-complete problem. The result still holds if P is globally consistent and d assigns only one variable.*

Proposition 13 *Let $\langle X, D, C \rangle$ be a globally consistent CSP, $x \in X$ and $v \in D(x)$. Unless $P = NP$, there is no polynomial algorithm that provides, for any assignment d such that $vars(d) = \{x\}$ and $d(x) = v$, a restriction D' of D (i.e. for any $y \in X, D'(y) \subseteq D(y)$) such that $\langle X, D', C \rangle$ is globally consistent w.r.t. the assignment d .*

The formalization and complexity analysis of the question of the productions of conflicts have been studied e.g. in [2], assuming that contexts contains only elementary expressions (fluents). Their results extend straightforwardly to any kind of expression.

Definition 14 [2] Given a CSP $P = \langle X, D, C \rangle$ and a set Context of consistent boolean expressions on the variables of X , a conflict is a subset $R \subseteq \text{Context}$ such that $\langle X, D, C \cup R \rangle$ is an inconsistent CSP. It is minimal iff there is no subset conflict such that $R' \subsetneq R$.

Proposition 15 [2] Given a consistent CSP $\langle X, D, C \rangle$, a set Choices of boolean expressions on the variables of X , and a set $R \subseteq \text{Choices}$, deciding R is a minimal conflict is a BH_2 -complete problem. The result still hold when Choices is a set of fluents.

The question of minimal price assessment can be easily formalized thanks to the notion of soft CSP (see e.g. [4]), extended to the case of negative costs. Recall that given a totally ordered set L , a soft constraint (or "cost function") is a function s mapping each element of $D(\text{vars}(s))$ to a value in L , generally $L = \mathbb{Z}^+ \cup \{+\infty\}$. Classically, the elements in L are combined by a binary operation $\oplus$ that is associative, commutative and monotonic, and equipped with an identity element. Here, we relax the assumption of monotony and let the costs being any integer values (negative costs are possible). In the present application, we are looking for the minimal prices of the vehicles compatible with the current assignment.

Proposition 16 Given a consistent CSP $\langle X, D, C \rangle$, a set S of cost functions on X taking its values in $\mathbb{Z}$, a partial assignment d , and an integer $\alpha \in \mathbb{Z}$, deciding whether there exist a $d' \in \text{Sols}(P)$ such that $d' \models d$ and $\sum_{s \in S} s(d') < \alpha$ is a NP-complete problem.

The question of the delay assessment can be modeled in a simpler way, using a distinguished variable x_t and representing the data relative to the delay by boolean expressions of the form: $c_i \implies x_t \in UI_i$, UI_i being a union of intervals and c_i a boolean expression (eg: $\text{color} = \text{green} \implies x_t \in (-\infty, 8] \cup [10, +\infty)$).

Proposition 17 Given a consistent CSP $P = \langle X, D, C \rangle$, a positive integer variable $x_t \in X$ such that each constraint bearing on x_t is of the form $c_i \implies x_t \in UI_i$ (c_i being a boolean expression and UI_i a union of intervals), d a (partial) assignment and a positive integer α , deciding whether there exists an assignment $d' \in \text{Sols}(P)$ such that $d' \models d$ and $d'(x_t) < \alpha$ is a NP-complete problem. The result still holds when each constraint of C bears on x_t and each UI_i contains only one interval

The completion of the configuration under a criteria of minimization of the price or the delay is obviously a variant of the previous one: finding an complete assignment d' such that $d' \in \text{Sols}(P)$, $d' \models d$ and d' minimizes the price/the delay is thus a NP-hard problem. The completion of the configuration without any criterion is probably not easier, even under the assumption that the CSP is globally consistent: we know that there exists $d' \in \text{Sols}(P)$ such that $d' \models d$, but we do not know how to get it. The conjecture is that, unless $P=NP$, there is no polynomial algorithm providing a solution to any globally consistent problem.

5 Conclusion

Through the study of a series of business tasks exploiting a constraint-based modeling of a vehicle range, it appears that configuration problems call for much more requests than those usually studied. The search for an (optimal) solution satisfying all the constraints is far from being the only request : the notion of global consistency, in different variants, play for instance a central role ; other

meaningful requests include not only counting problems, but also inference and marginalization problems that are sometimes beyond NP but stay in the first levels of the polynomial hierarchy.

For most of the tasks, the CSP containing the basic instance is highly consistent (it contains millions of solutions), known in advance, and is exploited by several different tasks. That is why the Applied Artificial Intelligence Department which is in charge of these applications in the Renault company, have oriented its developments toward an approach of the problem by knowledge compilation [14], that proves to be efficient (e.g. about 10^{-2} seconds for the problem of interactive on-line configuration). The question is now to determine (1) whether the algorithms promoted by the CSP community can be used in this context with a comparable efficiency, and (2) whether this community can provide efficient algorithms for problems that strongly differ from the search for an (optimal) solution, e.g. the marginalization of the CSP.

REFERENCES

- [1] M. Aldanondo, H. Fargier, and M. Veron, 'From csp to configuration problems.', in *AAAI-1999 Workshop on Configuration*, pp. 101–106., (1999).
- [2] J. Amilhastre, H. Fargier, and P. Marquis, 'Consistency restoration and explanations in dynamic CSP - application to configuration', *Artificial Intelligence*, **135**(1-2), 199–234, (2002).
- [3] A. Chmeiss, V. Krawczyk, and L. Sais, 'Redundancy in csp's', in *ECAI*, pp. 907–908, (2008).
- [4] M. C. Cooper and T. Schiex, 'Arc consistency for soft constraints', *Artificial Intelligence*, **154**(1-2), 199–227, (2004).
- [5] Johan de Kleer, 'Problem solving with the atms', *Artificial Intelligence*, **28**(2), 197–224, (1986).
- [6] J. L. de Siqueira N. and J-F Puget, 'Explanation-based generalisation of failures', in *ECAI*, pp. 339–344, (1988).
- [7] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', in *ECAI*, pp. 146–150, (2000).
- [8] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner, 'Configuring large-scale systems with generative constraint satisfaction', *IEEE Intelligent Systems, Special Issue on Configuration*, **13**(4), (July 1998).
- [9] E. Gelle and R. Weigel, 'Interactive configuration using constraint satisfaction techniques', in *PACT-96*, pp. 37–44, (1996).
- [10] U. Junker and D. Mailharro, 'The logic of ilog (j)configurator: Combining constraint programming with a description logic', in *Proc. IJCAI-03 Workshop on Configuration*, pp. 13–20, (2003).
- [11] U. Junker and D. Mailharro, 'Preference programming: Advanced problem solving for configuration', *AI EDAM*, **17**(1), 13–29, (2003).
- [12] Ulrich Junker, 'Quickxplain: Conflict detection for arbitrary constraint propagation algorithms', in *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, pp. 81–88, (2001).
- [13] S. Mittal and B. Falkenhainer, 'Dynamic constraint satisfaction problems', in *AAAI*, pp. 25–32, (1990).
- [14] B. Pargamin, 'Vehicle sales configuration: the cluster tree approach', in *ECAI'02 Configuration Workshop*, (2002).
- [15] D. Sabin and E. C. Freuder, 'Configuration as composite constraint satisfaction', in *AI and Manufacturing Research Planning Workshop*, pp. 153–161, (1996).
- [16] T. Schiex, 'Possibilistic constraint satisfaction problems or "how to handle soft constraints?"', in *UAI*, pp. 268–275, (1992).
- [17] T. Soiminen, E. Gelle, and I. Niemela, 'A fixpoint definition of dynamic constraint satisfaction', in *Proceedings of CP'99*, pp. 419–433, (1999).
- [18] M. Stumptner and A. Haselböck, 'A generative constraint formalism for configuration problems', in *Proceedings of the Third Congress of the Italian Association for Artif. Int. (AI*IA)*, 302–313, (1993).
- [19] M. Veron and M. Aldanondo, 'Yet another approach for ccsp for configuration problem', in *Proc. ECAI-2000 Workshop on Configuration*, pp. 59–62, (2003).

Annex: Sketches of the proofs

Sketch of proof [Proposition 1] The membership to NP is obvious⁴. The NP-hardness is get by reduction of SAT-CNF: X contains all the boolean variables of the CNF plus an additional boolean variable x , and C contains as many constraint $x = \top \implies cl_i$ as the number of clauses cl_i in the CNF. Obviously, the $\langle X, D, C \rangle$ is consistent and the CNF is satisfiable iff $\langle X, D, C \cup \{(x = \top)\} \rangle$ is consistent. $\square$

Sketch of proof [Proposition 3] The proof of BH_2 -hardness is inherited from [2]: this paper indeed considers contextual conflicts built on unary constraints, i.e. on fluents. The membership to BH_2 is easy to show: showing the inconsistency of $\langle X, D, C \cup C' \rangle$ belongs to CO-NP. To show that C' is minimal, for any $c \in C'$, consider the set $C'' = C' \setminus \{c\}$ and show that $\langle X, D, C \cup C'' \rangle$ is consistent (using a NP-oracle). $\square$

Sketch of proof [Proposition 5] Reduction from SAT-CNF: X contains all the propositional variables of the CNF and an additional variable x ; C contains as many constraints $x = 1 \implies cl_i$, as the number of clauses cl_i in the CNF; C moreover contains the constraint $c: x \in \{1, 2\}$. P is consistent (just set $x = 2$). c is globally consistent iff the CNF is satisfiable.

The membership to NP can be easily proved assuming that the number of variables in $vars(c)$ is bounded: list the assignments $d \in D(vars(c))$ such that $d \models c$, use a NP-oracle to guess a d' , and check $d' \in Sols(P)$ and that $d' \models d$. $\square$

Sketch of proof [Proposition 6 (validity of a description)] Simply reduce the problem of clausal inference, P being the CNF and c the clause the inference of which is to be shown. The membership to CO-NP obvious, considering the co-problem: let the NP-oracle guess a d , check that it satisfies P then that it does not satisfy c . $\square$

Sketch of proof [Proposition 7 (projection)] For any set of variables $Y \subseteq X$, c can be the condition $\bigwedge_{x \in Y} (x, D(x))$: this loose condition "selects" some variables without constraining their values. Hence, c is equivalent to $Sols(\langle X, D, C \rangle)^{\downarrow vars(c)}$ iff, for any assignment of Y , there exist an assignment of $X \setminus Y$ that satisfies the CSP. The canonical problem of Π_2^P ($\forall Y_1, \exists Y_2 \Sigma$, Σ being CNF) can thus be polynomially reduced to the test of equivalence between the projection on Y_2 of a CSP representing the CNF and the condition $c = \bigwedge_{x \in Y_2} (x, \{true, false\})$. Proving the membership to Π_2^P is easy. The condition $d \in Sols(\langle X, D, C \rangle)^{\downarrow vars(c)} \implies c(d) = \top$ can be falsified using one NP-oracle (guess d , check that $d \models P$ and that $d \not\models c$). In order to falsify $c(d) = \top \implies d \in Sols(\langle X, D, C \rangle)^{\downarrow vars(c)}$, let the oracle guess an assignment of $vars(c)$, check that $d \models c$, and that d cannot be extended to a solution of P (this subproblem belongs to CO-NP). $\square$

Sketch of proof [Proposition 10 (Concision, exhaustivity, exclusivity)] In order to prove the hardness of these requests, we use the following reduction of the SAT-CNF problem: X contains all the Propositional variables of the CNF and as many variable x_i as the number of clauses of the CNF; the constraints in C are of the form $x_i \leftrightarrow cl_i$, where cl_i is a clause of the CNF; this CSP is obviously consistent, and testing the satisfiability of the original CSP is equivalent to testing the consistency of the CSP with the assignment $c': x_1 = \top \wedge \dots \wedge x_n = \top$. Let $C' = \{c'\}$: the CNF is satisfiable iff C' is concise. $C' = \{\neg c'\}$ the CNF is unsatisfiable iff C' is exhaustive. Let $c': x_1 = \top$ and $c'': x_2 = \top \wedge \dots \wedge x_n = \top$. The

CNF is unsatisfiable iff $C' = \{c', c''\}$ is mutually exclusive.

The membership to NP (for the problems of concision) or to CO-NP (for the exhaustivity and exclusivity requests) is obvious. $\square$

Sketch of proof [Proposition 12 (global consistency)] Consider a CNF. Build the CSP $P = \langle X, D, C \rangle$ such that X contains all the variables of the CNF plus two additional boolean variables, say $x_\top$ and x_2 . C contains as many constraints $x_\top = \top \wedge x_2 = \top \implies cl_i$ as the number of clauses cl_i in the CNF. Consider that $d(x_\top) = \top$ and $\forall x \in X \setminus \{x_\top\}, d(x) = *$. The problem is globally consistent and for any boolean variable $y \neq x_\top, x_2$, there is a solution d' such as $d'(y) = \top$ and $d' \models d$ and another such that $d'(y) = \perp$ and $d' \models d$ (just set $d'(x_2) = \perp$). So, the CSP is globally consistent w.r.t. d iff there is a solution d' such as $d'(x_2) = \top$, i.e. iff the CNF is satisfiable.

Membership to NP: for any x and any v , guess d' , check that $d' \in Sols(P)$ and $d' \models d$. $\square$

Sketch of proof [Proposition 13 (Incremental global consistency)] Consider the CSP of the previous proof. If there were a polynomial algorithm providing computing $\langle X, D', C \rangle$ globally consistent with the assignment $d(x_1) = \top, d(x) = *, \forall x \neq x_1$, we would be able to decide in polytime whether the problem admits a solution d' such that $d'(x_2) = \top = d'(x_1)$, i.e. whether the CNF is satisfiable. Which does not hold unless $P = NP$. $\square$

Sketch of proof [Proposition 15 (Conflicts, restaurations)] See [2] for the case *Choices* being a set of fluents. The difficulty is not increased by *Choices* being a set of consistent boolean expressions. $\square$

Sketch of proof [Proposition 16 (Assessment of the price)] The problem of minimization indeed encompasses the question of consistency of a CNF formula (letting d being the empty assignment and using the same transformation that for classical soft CSPs (satisfying a clause has a cost of 0, violating it a cost of 1 and $\alpha = 0$)). $\square$

Sketch of proof [Proposition 17 (Assessment of the Delay)]

Minimal Delay assessment: The problem is NP-hard when d does not assign any variables and each of the delay constraints is of the form $c_i \implies x_t \in [a_i, +\infty)$: we shall then encode a problem of optimization in a possibilistic CSP [16] (or a CNF of possibilistic logic), where the aim is to minimize the priority of the most important of the violated constraint: each constraint cl_i of priority α_i in the possibilistic base is translated into a delay constraint $\neg cl_i \implies x_t \in [\alpha_i, +\infty)$. The membership to NP is obvious. $\square$

⁴ The proofs of membership as well as the proofs of polynomiality of the reductions are generally obvious, and thus skipped for the sake of brevity

Modeling Technical Product Configuration Problems

Andreas Falkner and Alois Haselböck and Gottfried Schenner

Siemens AG Österreich, Austria, email: {andreas.a.falkner,alois.haselboeck,gottfried.schenner}@siemens.com

Abstract. This paper describes and evaluates different approaches for modeling technical product configuration problems. Several languages from various existing modeling paradigms - like logic-based, constraint-based, or object-oriented - are used on a typical example. The aim is not to improve or exhaust these techniques, but to test their applicability on configuration problems and to discuss their advantages and limitations.

1 INTRODUCTION

Product configurators have a long history in artificial intelligence (see [1], [2]), the first and most famous example being the rule based configurator R1/XCON system [3] for DEC-Computer. Nowadays several vendors of commercial configurators based on AI methods are established (SAP, Oracle, ILOG, Tacton, Configit etc.). Nevertheless, many products especially in technical domains are configured by engineers using in-house software without AI technology. A reason for this may be that there is little literature available on how to use AI methods specifically for product configuration.

This paper intends to improve this situation. In the rest of the paper we show the modeling of a typical configuration problem with different languages. As an example we use a fictitious people counting system for museums. The structure of the problem is similar to problems we encountered in different real-world domains of our technical product configurators: to configure a technical product by selecting, arranging and parameterizing components from a set of predefined component types.

To model the example, we have chosen a set of non-commercial AI languages, suitable for knowledge-based systems, which cover a wide range of different paradigms:

- UML/OCL (object-oriented)
- OWL (description logics)
- Alloy and DLV (logic-based)
- GCSP (constraint-based)
- CHR (constraint logic programming)
- Graph partitioning (mathematical)

2 PROBLEM DESCRIPTION

A fictitious museum has lots of rooms and doors between them. In order to prevent damage to the objects in exhibition, the number of visitors shall be restricted. This is done by a people counting system which consists of following components: door sensors, counting zones, and communication units.

A door sensor detects everybody who moves through its door (directed movement detection). There can be doors without a sensor.

Any number of rooms may be grouped to a counting zone. Each zone knows how many persons are in it (counting the information from the sensors at doors leading outside of the zone). Correct function requires that all doors leading outside a zone have a sensor (the corresponding constraint is not part of this problem). Zones may overlap or include other zones, i.e. a room may be part of several zones.

A communication unit can control at most two door sensors and at most two zones. If a unit controls a sensor which contributes to a zone on another unit, then the two units need a connection: one is a partner unit of the other and vice versa. Each unit can have at most N partner units. For the sake of simplicity, we use $N=2$ throughout this paper, whereas higher values for N are more common in real-world problems of this kind. Of course, the problem diminishes or even vanishes when N is chosen sufficiently high or unbounded, but we assume technical reasons inhibiting high values.

PartnerUnits problem: Given a consistent configuration of door sensors and zones, find a valid assignment of communication units (i.e. with max. 2 partners).

Example 1: Rooms 1 to 8 with eleven doors, eight of them having a door sensor, e.g. doors between rooms 1 and 2, or 3 and 4, but not between 2 and 3.

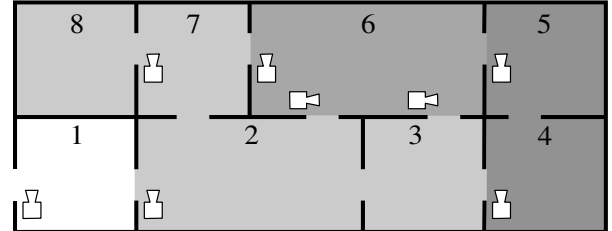


Figure 1. Room layout of example 1

Given seven zones named by the rooms which they contain: Z1 (white), Z2378 (light gray), Z45 (dark grey), Z6 (medium grey), Z456, Z2367, Z2345678. They are consistent to the door sensors because all doors without sensors are only inside zones. The sensor between 7 and 8 is ignored for Z2378 and Z2345678, but necessary for Z2367.

The eight door sensors get the names D01, D12, D26, D34, etc. by the rooms which they connect (and 0 for the outside, respectively).

The relation between zones and door sensors can be represented as a bi-partite graph (see Figure 2). An instance of a solution using only 4 units is shown in Table 1.

The problem has several interesting properties. Although in its original form it is given as a constraint satisfaction problem (i.e. finding any solution), it can also be seen as a constraint optimization problem: find a solution with the minimal number of units (or even with the minimal number of partner units). The absolute minimum is the smallest integer greater or equal to the half of the

maximum of the numbers of zones and door sensors. However, we have no proof whether a solution with that minimum exists (given that a solution exists at all). As in real-world a minimal solution is important, the original problem may be rephrased to finding a solution with a given number of units (i.e. the absolute minimum).

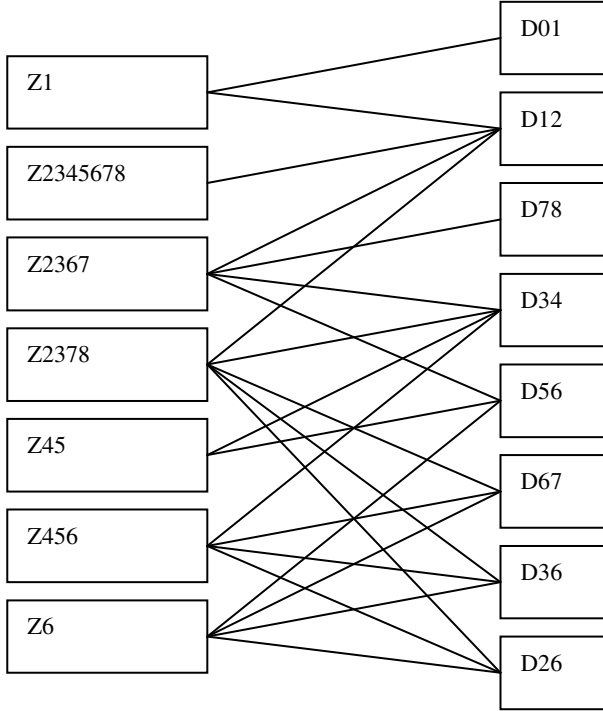


Figure 2. Relation between zones and door sensors of example 1

Unit	Zones		Door Sensors		Partner Units	
U1	Z1	Z2345678	D01	D56	U3	U4
U2	Z2378	Z456	D34	D67	U3	U4
U3	Z45	Z6	D26	D36	U1	U2
U4	Z2367	-	D12	D78	U1	U2

Table 1: A minimal solution of example 1

For small examples (i.e. less than 6 sensors and 6 zones), it is easy to find a solution. The same is true when the connection degree is small (e.g. the number of sensors for each zone is less than 3 and vice versa). For bigger configurations however, the constraint of maximal $N=2$ partner units makes the problem hard.

The problem is well suited for comparing different solution approaches because structurally different problem instances can be generated easily, varying the number of zones and sensors, the degree of connectivity, and the maximum of allowed partner units (i.e. $N>2$).

3 MODELING

A model mainly consists of the representation of the configuration components as well as constraints and rules defining valid solutions. For many technical domains, the models get complex and large, so that a high-level modeling language is required. It should provide for an easy, natural and elegant problem description, supporting readability, validation and maintainability of the model.

We demonstrate the modeling of the PartnerUnits problem prototypically by different languages of some prominent knowledge representation paradigms.

3.1 UML/OCL

UML class diagrams are a common way to describe the structure of a system in object-oriented modeling. The primary use of UML diagrams is to communicate the model visually inside a software project. In combination with OCL (Object Constraint Language) it is also expressive enough to describe product configuration [4].

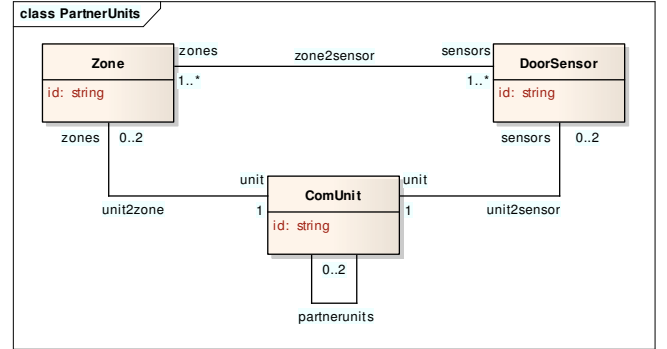


Figure 3. UML diagram of PartnerUnits problem

The UML diagram shows a class diagram derived from the description. It does not represent rooms as they are not part of the concrete problem at hand. For all associations, it contains the cardinality constraints (minimal/maximal number of connected elements), but there is no way to express the fact that the *partnerunits* association is derived from the path over the *zone2sensor* relation inside the class diagram. It must be expressed with an OCL constraint:

```
context ComUnit inv:
myPartnerUnitsSensor =
  sensors.zones.unit->excluding(self)->asSet()
and myPartnerUnitsZone =
  zones.sensors.unit->excluding(self)->asSet()
and myPartnerUnitsSensor
  ->union(myPartnerUnitsZone)->size() <= 2
```

Although UML/OCL is widely used in software engineering projects especially for the MDA (model driven architecture) approach, there are few tools available that actually support reasoning with UML/OCL.

One example is the UML-based specification environment USE [5]. USE (<http://www.db.informatik.uni-bremen.de/projects/USE/>) allows the creation of example configurations (called snapshots in USE-terminology). The validity of the examples can be checked in relation to the UML/OCL specification. Unfortunately it is not possible to automatically generate instantiations of the specification i.e. find solutions.

3.2 Description Logics

Description logics [6] is widely used for formal representation and reasoning over complex knowledge networks, like for semantic systems. For years it has been used for configuration tasks as well [7], the first reported application being AT&T's PROSE system

(see McGuinness' chapter on Configuration in [6]). Due to its wide distribution, there are several tools and representations. We chose Protégé (<http://protege.stanford.edu/>) and Manchester OWL syntax for specifying an ontology for the PartnerUnits problem. It is straight-forward to model the concepts (classes) and their relations (object properties):

```
Class: Zone
Class: DoorSensor
Class: ComUnit

ObjectProperty: zone2sensor
  Domain: Zone
  Range: DoorSensor
  InverseOf: sensor2zone

ObjectProperty: sensor2zone
  Domain: DoorSensor
  Range: Zone
  InverseOf: zone2sensor

ObjectProperty: zone2unit
  Domain: Zone
  Range: ComUnit
  InverseOf: unit2zone

ObjectProperty: unit2zone
  Domain: ComUnit
  Range: Zone
  InverseOf: zone2unit

ObjectProperty: sensor2unit
  Domain: DoorSensor
  Range: ComUnit
  InverseOf: unit2sensor

ObjectProperty: unit2sensor
  Domain: ComUnit
  Range: DoorSensor
  InverseOf: sensor2unit

ObjectProperty: partnerunits
  Characteristics: Symmetric, Irreflexive
  Domain: ComUnit
  Range: ComUnit
```

Their cardinalities are restricted by numeric constraints:

```
Class: ZoneConstrained
  SubClassOf:
    Zone,
    zone2sensor min 1 DoorSensor,
    zone2unit exactly 1 ComUnit

Class: DoorSensorConstrained
  SubClassOf:
    DoorSensor,
    sensor2zone min 1 Zone,
    sensor2unit exactly 1 ComUnit

Class: ComUnitConstrained
  SubClassOf:
    ComUnit,
    unit2sensor max 2 DoorSensor,
    unit2zone max 2 Zone,
    partnerunits max 2 ComUnit
```

The *partnerunits* relation must equal the path via *zone2sensor* (the inverse direction and the exclusion of self as a partner are covered by the specified characteristics *symmetric* and *irreflexive*):

```
Class: ComUnitProperlyConnected
  SubClassOf:
    ComUnitConstrained,
    unit2zone some (zone2sensor some (sensor2unit
      only (partnerunits some ComUnit))),
    partnerunits only (unit2zone some (zone2sensor
      some (sensor2unit some ComUnit)))
```

An alternative to the latter sub-class would be to compute the *partnerunits* by an external function (as an extension to pure OWL).

The notation mirrors the UML view well, although users less fluent in DL might be puzzled by the representation of cardinality constraints as *SubClassOf* and by the variable-free quantifications.

In general, available reasoners cope better with classification (i.e. subsumption of sub-classes) than with checking instances (model checking) or finding solutions (model finding). Performance for big configurations has yet to be evaluated.

3.3 Alloy

Alloy (<http://alloy.mit.edu/alloy4/>) is a lightweight specification language and tool [8]. The language of Alloy, which is a combination of first-order logic and relational calculus, is relatively easy to learn and use (compared to other specification languages). Using the Alloy Analyzer tool, instances satisfying the specification can be found and assertions about the specification can be checked within a given scope. An Alloy specification of the PartnerUnits problem looks like this:

```
module PartnerUnits

sig Zone {
  zone2sensor: set DoorSensor
}
sig DoorSensor {}
fact cardinalities_zone2sensor {
  all z:Zone | #z.zone2sensor > 0
  all d:DoorSensor | #d.~zone2sensor > 0
}
sig ComUnit {
  unit2sensor: set DoorSensor,
  unit2zone: set Zone,
  partnerunits: set ComUnit
}
fact cardinalities_unit2sensor {
  all u:ComUnit | #u.unit2sensor <=2
  all d:DoorSensor | #d.~unit2sensor = 1
}
fact cardinalities_unit2zone {
  all u:ComUnit | #u.unit2zone <=2
  all z:Zone | #z.~unit2zone = 1
}
fact derivedassoc_partnerunits {
  partnerunits =
    unit2zone.zone2sensor.~unit2sensor +
    ~(unit2zone.zone2sensor.~unit2sensor) -
    iden
}
fact cardinalities_partnerunits {
  all u: ComUnit | #u.partnerunits <= 2
}
```

The *sig* definitions of the Alloy specification correspond to the UML class definitions. An expression like *zone2sensor* denotes the binary relation between *Zone* and *DoorSensor*. The symbol " $\sim$ " denotes the inverse of a relation i.e. $\sim$ *zone2sensor* is the relation from *DoorSensor* to *Zone*. The dot operator denotes the relational join, e.g. a navigational expression like *unit2zone.zone2sensor* evaluates to a binary relation that relates the units to all sensors that belong to a zone of the unit.

Alloy can be used to check assertions about the specification. For instance, it is easy to conclude that any configuration containing a zone with more than 6 door sensors is inconsistent. We can prove this assumption (within the given scope) by checking the following assertion in Alloy:

```
check { all z: Zone | #z.zone2sensor <= 6 }
for 10 but 5 int
```

Alloy tries to find a counterexample and does not succeed, because the assertion is valid. Since the problem is symmetric for zones and sensors, there can not be more than 6 zones for a sensor as well. Therefore the cardinalities of both sides of the *zone2sensor* association can be restricted to 1..6 (from 1..*). Deriving cardinality restrictions from an UML model is an area of active research [9]. Such restrictions are very valuable for ruling out inconsistent requirements (like in the classical pigeon-hole problem) at an early stage of the configuration process.

3.4 Datalog (DLV)

DLV-Complex (<http://www.mat.unical.it/dlv-complex>) is an Answer Set Programming System extending DLV (<http://www.dlvsystem.com/>), a system for disjunctive datalog with constraints, true negation and queries [10]. It offers a very concise representation of the problem.

The relation between zones and door sensors and the maximally usable amount of communication units are given as positive facts. The implicit unique name assumption (UNA) ensures that the listed zones and door sensors are considered different, e.g. for a simple example with two zones *z1* and *z2* and four door sensors *d1* to *d4* as shown in Figure 4.

Zone *z1* is associated with *d1*, *d2*, and *d3*. Zone *z2* is associated with *d3* and *d4*. We need at least $\max(3,4)/2 = 2$ units:

```
zd(z1,d1). zd(z1,d2). zd(z1,d3).
zd(z2,d3). zd(z2,d4).
unit(1..2).
```

We formulate the possible assignment of units to zones and sensors as a disjunction of positive and negative facts. Constraints restrict their cardinalities: not more than 2 zones per unit, exactly one unit per zone (analogous for sensors):

```
zu(Z,U) v ~zu(Z,U) :- zd(Z,_), unit(U).
:- unit(U), 2 < #count{Z: zu(Z,U)}.
:- zd(Z,_), 1 < #count{U: zu(Z,U)}.
:- zd(Z,_), 1 > #count{U: zu(Z,U)}.
du(D,U) v ~du(D,U) :- zd(_,D), unit(U).
:- unit(U), 2 < #count{D: du(D,U)}.
:- zd(_,D), 1 < #count{U: du(D,U)}.
:- zd(_,D), 1 > #count{U: du(D,U)}.
```

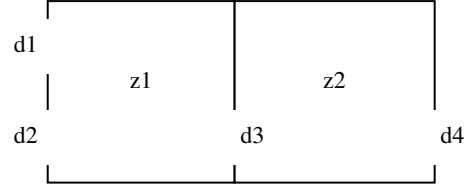


Figure 4. Simple example with 2 zones and 4 door sensors

Similarly, we calculate and restrict the partner units:

```
pu(U,P) :- zu(Z,U), zd(Z,D), du(D,P), P!=U.
pu(U,P) :- pu(P,U).
:- unit(U), 2 < #count{P: pu(U,P)}.
```

Running the program - and filtering the relevant positive facts *zu*, *du*, and *pu* - results in:

```
zu(z1,1). zu(z2,1).
du(d1,1). du(d2,1). du(d3,2). du(d4,2).
pu(1,2). pu(2,1).
```

The notation is very concise, although one has to get used to the constraints (" $\sim$ ") describing what is not allowed. Similar for the idea/pattern to specify assignment with the disjunction of positive and negative facts.

The specification can be used for model finding. It performs equally for problems with lower and higher connection degree, quite well for smaller problems like the example at the beginning of this paper.

Furthermore it will not find minimal solutions if there are too many units available. One can avoid this by setting the number of available units to the minimum - e.g. *unit(1..2)* in the example above - and iteratively increase the number when necessary.

The use of weak constraints (" $\sim$ ") for optimizing the solution (in case the number of available units is higher than the number of necessary ones) is elegant but increases the runtime considerably:

```
used(U) :- zu(_,U).
used(U) :- du(_,U).
:~ used(U).
```

3.5 Generative Constraint Satisfaction

Constraint satisfaction is widely used to represent and solve configuration problems. A CSP in the classical sense consists of a fixed set of variables and their domains, as well as constraints which restrict the assignment of the variables. A valid solution is an assignment of all variables with values from their domains where all constraints are satisfied. Because of the dynamic nature of most configuration problems, a formulation of the PartnerUnits problem as a static CSP may not be the best way of meeting the objectives. As an example of an extended CSP schema, we use Generative CSP [11].

The classes (i.e. component types) of our problem are *Zone*, *DoorSensor* and *ComUnit*. Each class represents a theoretically infinite set of instances (i.e. components).

A class can have attributes, associations and constraints. Whenever a new instance of a class is created, it gets instances of its attributes, associations and constraints, too. This is the object-oriented view of the modeling.

From the constraint-oriented point of view, attributes and associations represent the variables. The domain of an attribute variable is its type, e.g. Boolean, an integer interval, or a symbol enumeration. Associations are bi-directional and induce two association variables, one for each side. The domain of such an association variable is the set of all instances of the class specified on the other side of the association. E.g. the association definition

```
assoc: Zone.units(1) - ComUnit.zones(0..2)
```

represents the connection of zones to units. The two association variables induced are *Zone.units* (the link from a zone to its unit) and *ComUnit.zones* (the link from a unit to all its associated zones). Allowed cardinalities are given in brackets. Implicit constraints check that all instances associated to an association variable are of the specified type (e.g. *ComUnit* for *Zone.units*) and that the given cardinalities are not violated (e.g. *Zone.units* must contain exactly one instance).

All the associations of our PartnerUnits problem are:

```
assoc: Zone.sensors(*) - DoorSensor.zones(*)
assoc: Zone.unit(1) - ComUnit.zones(0..2)
assoc: DoorSensor.unit(1) - ComUnit.sensors(0..2)
assoc: ComUnit.partnerunits(*) -
    ComUnit.partnerunits(*)
```

In fact, these class and association definitions represent the UML diagram of Fig. 3 in a straight-forward and compact way. The interesting parts are the following constraint definitions for units.

```
class ComUnit:
    constraint:
        partnerunits = this.zones.sensors.unit
                    + this.sensors.zones.unit
                    - this
    constraint:
        partnerunits.cardinality <= 2
```

The first constraint specifies which elements are to be in the partner association of each unit. These are all units reachable via its zones and its sensors, where the unit itself is not member of the association. (Notational remark: *this.zones* represent a set. *this.zones.units* represents the union of all *z.units* for all *z* of *this.zones*.)

The second constraint simply states that there are only 2 partner units allowed for each unit.

Given in our PartnerUnits problem is the set of all zone and sensor instances and the associations between them. Wanted are the instances of units along with their associations to zones, sensors and partner units so that no constraint is violated.

Generative CSP is well-suited for the modeling of configuration problems because of its object-oriented approach (natural and maintainable formulation of the problem structure), its constraint-orientatedness (declarative formulation of the problem logics), and its dynamicity. Suitable solvers (e.g. backtracking, heuristic repair) can easily be integrated.

3.6 CHR

Constraint Handling Rules (CHR) is a declarative concurrent committed-choice constraint logic programming language consisting of guarded rules that transform constraints (represented

as multisets of relations) until no more change occurs [12]. With its built-in reasoning mechanism for simplification and propagation rules it is well suited for optimizing constraint satisfaction.

We use CHR with host language SWI-Prolog (<http://www.swi-prolog.org/>). Therefore we can take advantage of Prolog's inference machine and variable unification: The relation between zones and door sensors as an input is represented by facts *zdu/2* which relate variables which later will be instantiated to the unit for that zone or sensor (exploring an idea of [13]), e.g. for the simple example in section 3.4:

```
?- zdu(Z1,D1), zdu(Z1,D2), zdu(Z1,D3),
    zdu(Z2,D3), zdu(Z2,D4),
    label([z1-Z1,z2-Z2],[Z1,Z2],
        [d1-D1,d2-D2,d3-D3,d4-D4],[D1,D2,D3,D4],
        [1,2]).
```

They are simplified to relations for the partner units (*pu/2*):

```
zdu(ZU,DU) <=> nonvar(ZU), nonvar(DU)
                | pu(ZU,DU), pu(DU,ZU).
```

Partner units are optimized and restricted (to 2):

```
% same unit is never a partner
pu(U,U) <=> true.
% remove duplicates
pu(U,PU) \ pu(U,PU) <=> true.
% not more than 2 partner units
pu(U,_), pu(U,_), pu(U,_) <=> fail.
```

The placement of doors and sensors to units is done by a naïve labeling of zones and sensors with a unit, which at first tries to place two zones and two sensors onto one unit, and only if it fails, places fewer ones. By using variable binding for that, it realizes a natural way of symmetry breaking (in addition to the explicit symmetry breaking in the second line below). If a variable is bound then it triggers generation of the partner unit in the simplification rule of *zdu/2*.

```
label(Zs,ZVs,Ds,DVs,[U|Us]) <=>
    (ZVs=[U|ZVs1]; ZVs=[],ZVs=ZVs1), % symm. break.
    (select(U,ZVs1,ZVs2); ZVs1=ZVs2),
    (select(U,DVs,DVs1); DVs=DVs1),
    (select(U,DVs1,DVs2); DVs1=DVs2),
    label(Zs,ZVs2,Ds,DVs2,Us).
label(_,_,_,[_|_],[]) <=> fail. % too few units
label(_,[_|_],_,[]) <=> fail.
label(Zs,[],Ds,[],_) <=> label(Zs,Ds).
```

The results are the facts for the partner units and the final labeling, e.g.:

```
label([z1-1,z2-1],[d1-1,d2-1,d3-2,d4-2]),
pu(1,2), pu(2,1).
```

The notation is logic programming and therefore clear for anyone knowing some Prolog. However, formulating or even understanding expected reasoning results requires deep insight into the operational semantics of CHR.

The specification can be used for model finding and performs similar to a traditional backtracking approach.

3.7 Graph partitioning

The PartnerUnits problem can also be described as a special variant of graph partitioning.

Given a bipartite graph $B = (V_1; V_2; E)$, a partnerunits graph for B is a graph $P = (V_p; E_p)$, where

- V_p is a partitioning of $V_1 \cup V_2$ such that each set in V is non-empty and contains at most 2 nodes from V_1 and at most 2 nodes from V_2 and
- $(c_1; c_2) \in E_p$ iff $(b_1, b_2) \in E$ for some $b_1 \in c_1$ and some $b_2 \in c_2$.

Problem: *N-partnerunits graph*

Instance: A bipartite graph B .

Question: Is there a partnerunit graph for B with node degree at most N ?

Graph partitioning is a well known NP-complete problem for which many heuristic algorithms exist [14]. In contrast to the standard graph partitioning problem, where every edge is taken in consideration, in our problem only one edge for every connected partition is required. This is the reason why adaptations of the existing algorithms are necessary.

The formulation of the PartnerUnits problem with such a mathematical language relies on a well-established theory and problem-specific algorithms. But it is highly abstract and needs a deep mathematical background, which cannot always be expected from knowledge engineers. Furthermore, this kind of modeling usually is not very robust to changes. Often a slight modification in the problem description destroys the mapping to the mathematical problem formulation and makes a new examination and implementation necessary.

4 CONCLUSION

In this paper, we concentrated on modeling the PartnerUnits problem, not on finding solutions to it. This will be covered in a paper submitted to the forthcoming AI EDAM special issue on configuration.

Although it was possible to model the PartnerUnits problem with the discussed approaches, they all had a problem with large scale configurations (i.e. containing a lot of instances). Without a domain-specific heuristic or mapping the problem to algorithms from other fields (graph theory, OR etc.) these problems can't be solved. As stressed in [15] we can not expect one general method to solve all the problems of all domains.

Therefore we advocate the use of formal tools at an early stage of knowledge engineering to analyze the problem and its complexity before choosing a suitable solving technology.

What makes the PartnerUnits problem hard to solve is the restriction to N partner unit connections (e.g. $N=2$). It is worth asking the design engineers of the product whether this restriction is really necessary or whether there is another technical solution without this restriction. Experiences have shown that many hard configuration problems could be avoided just by an early integration of configurator architects into the product design process to make the product easier to configure [16].

At present, we do not see a well-established modeling language for product configuration although there are various suggestions over the last years, like [17]. The presented modeling examples show that all existing paradigms are more or less suited for an

understandable and concise representation of product configuration problems. Which one to choose depends on the familiarity of the knowledge engineer with the notation and on the expressive and computational power of language and tool applied to the given problem. Due to efficiency reasons for solving complex/large product configuration problems, often a reformulation of the model is necessary, which can only be done by highly experienced specialists for the selected modeling language.

An automatic transformation from an understandable high-level description to the specific best-performing solving framework (similar like some tools do already today with efficient SAT solvers) would be a great benefit for the "average" knowledge engineer, as long as transformation and back-transformation are transparent to the user.

REFERENCES

- [1] D. Sabin and R. Weigel, *Product Configuration Frameworks - A Survey*, IEEE Intelligent Systems 13(4), pp. 42-49, 1998.
- [2] A. Felfernig, *Standardized Configuration Knowledge Representations as Technological Foundation for Mass Customization*, IEEE Transactions on Engineering Management, 54(1), pp. 41-56, 2007.
- [3] J. McDermott, *RI: A rule-based configurator of computer systems*, Artificial Intelligence 19, pp. 39-88, 1982.
- [4] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, *Configuration knowledge representation using UML/OCL*, Proceedings of the 5th International Conference on the Unified Modeling Language. Springer-Verlag, 2002, pp. 49-62, 2002.
- [5] M. Gogolla, F. Büttner, and M. Richters, *USE: A UML-Based Specification Environment for Validating UML and OCL*, Science of Computer Programming, Vol. 69, pp. 27-34, 2007.
- [6] F. Baader, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, *The Description Logic Handbook*, Cambridge University Press, 2003.
- [7] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker, *Configuration knowledge representations for Semantic Web applications*, AI EDAM 17/1, pp. 31-50, 2003.
- [8] D. Jackson, *Alloy: a lightweight object modeling notation*, ACM Transactions on Software Engineering Methodologies 11, 2 (Apr. 2002), pp. 256-290.
- [9] A. Falkner, I. Feinerer, G. Salzer, and G. Schenner, *Computing Product Configurations via UML and Integer Linear Programming*, International Journal on Mass Customization (to appear), 2010.
- [10] T. Eiter, G. Gottlob, and H. Mannila, *Disjunctive datalog*, ACM Transactions on Database Systems 22/3, pp. 315-363, 1997.
- [11] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner, *Configuring large-scale systems with generative constraint satisfaction*, IEEE Intelligent Systems, 1998, 13(4), pp. 59-68, 1998.
- [12] T. Frühwirth, *Welcome to Constraint Handling Rules*, LNAI, Vol. 5388, 2008.
- [13] T. Frühwirth, Personal communication, 2009.
- [14] B.W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell system technical journal, 1970.
- [15] Z. Michalewicz and D.B. Fogel, *How to Solve It: Modern Heuristics*, Springer, 2004.
- [16] A. Falkner and A. Haselböck, *A Simple Evaluation Process for Configurability*, Proc. IJCAI-09 Workshop on Configuration, pp. 17-22, 2009.
- [17] T. Soinen, J. Tiihonen, T. Männistö, R. Sulonen, *Towards a general ontology of configuration*, AI EDAM 12, pp. 357-372, 1998.

A Generative Configuration Framework for Service Process Composition¹

Rajesh Thiagarajan and Wolfgang Mayer and Markus Stumptner²

Abstract. In our prior work we showed the benefits of formulating service composition as a Generative Constraint Satisfaction Problem (GCSP), where available services and composition problems are modeled in a generic manner and are instantiated on-the-fly during the solving process, in dynamic composition scenarios. In this paper, we (1) outline the salient features of our framework, (2) present extensions to our framework in the form of process-level invariants, and (3) share promising results of our experiments to evaluate the effectiveness of our framework in scenarios with varying difficulty.

1 Introduction

A vast number of proposals that exploit formal specifications of individual services to automatically select and compose individual services into executable service processes have been brought forward [17, 9, 7, 2, 11, 8]. While most frameworks can successfully address basic composition tasks, many are based on ad-hoc algorithms or lack a precise representation of a service's capabilities. Therefore, the problem of configuring and tailoring the software that implements a given service is often left aside.

Constraint satisfaction based configuration techniques have been proposed as an alternative to address these challenges in service composition scenarios, where both type and instance information of services and relevant data must be considered [6, 18, 5, 1]. Existing models compose services by treating services as components and assembling them. However, standard CSPs are insufficient to model configuration problems where the number of components to be configured is unknown. Existing CSP-based composition techniques handle this by pre-specifying the number or type of services to be composed. In general, such pre-specification is difficult to make since domain specific knowledge that is problem specific is not available apriori.

To address this gap, we present a generative consistency-based service composition approach that addresses these challenges [12]. We extend models that have been successfully applied to model and configure complex systems [19], to the software domain. The service composition problem is posed as a configuration task, where a set of service components and their interconnections are sought in order to satisfy a

given goal. Our framework is based on a declarative constraint language to express user requirements, process constraints, and service profiles on a conceptual and also on the instance level. We show that our meta-level constraint specifications permit to concisely represent conceptual information about processes, services and data structures involved in the composition. One of the major benefits of our approach is the provision to define problems independent of the number and type of services required, thereby overcoming the disadvantage of other models.

In this paper, we briefly outline our formalism that offers uniform (generic) constructs to represent service capabilities and semantics, represent data- and control flow between services (Section 2). We also present our extensions to the original model to unambiguously correlate conversations involving service instances or different parts of a workflow in a composition (Section 3). We also present the results from our experiments in which we evaluated our framework's effectiveness to handle non-trivial pessimistic scenarios, where service compositions often fail (Section 4). Experimental results indicate that even in such pessimistic settings our framework's performance is quite competitive to other composition systems.

2 GCSP-based Service Composition

Constraint Satisfaction Problem (CSP)³ is a successful problem solving technique applied in various domains including general planning [15, 10] and service composition [6, 18, 5, 1]. Generative CSPs (GCSPs) extend standard CSPs by lifting constraints and variables to a meta-level, where *generic constraints* are the primary modeling element. Generic constraints abstract the actual CSP variables into so-called *meta variables* that are instantiated into ordinary constraints over variables in a classical CSP during configuration. Generative configuration can be seen as the problem of incrementally extending a CSP network until all the generic constraints are satisfied. In our formalism, when a component (C) of a particular type is added to the configuration, related variables and constraints (instances of the generic constraints of C) are activated in the configuration process. This dynamic activation of CSP variables makes GCSPs suitable for dynamic situations where a priori prediction of the problem structure is difficult.

A GCSP is characterized by the set of available component types, their attributes and ports (to connect to other compo-

¹ This work was partially supported by the Australian Research Council (ARC) under grant DP0988961.

² Advanced Computing Research Centre, University of South Australia, {cirsrt,mayer,mst}@cs.unisa.edu.au

³ A CSP consists of a finite set of variables, set of values that each variable can be assigned to (the variable's *domain*), and a set of constraints restricting the valid assignments of values to variables.

nents), and a set of generic constraints. In generic constraints, meta-variables act as placeholders for component variables. Generic constraints can be seen as prototypical constraints on a meta-level that are instantiated into *ordinary* constraints over variables in the CSP. For example, assume a generic constraint $X \sqsubseteq \text{BudgetShipper} \Rightarrow X.\text{price} < 1000$ is given, stating the invariant that the value of attribute *price* for any data object of type *BudgetShipper* must be less than 1000. A generic constraint over a meta-variable is said to be consistent if and only if the constraint is satisfied for all instances over active CSP variables.

In [12], we extend the generative formalism in [4, 3] with the following features.

- We introduce connection components that act as connectors between services. The explicit representation of connectors provides a uniform interface contract between services and also serves as a means to model the provider-consumer relationship between services.
- A connection component also holds a representation of the data values that may be passed along the connection.
- To capture their semantics, we treat complex data objects as components. This facilitates the uniform handling of service and data components in the configuration, and has the additional benefit that generic constraints can be used to impose invariants on data structures throughout a configuration.
- We introduce non-local process level constraints to model data flow, control flow, and structural invariants of service processes.

Figure 2 exemplifies the structure of a service model in our framework. Individual service component and connectors are depicted as rectangular boxes, while data elements are represented as ovals. In order to facilitate reasoning about data objects attached to connections between service components, the connections are also modeled as components. Component constraints are formulated in terms of the local service component, the connection components that are directly connected to its ports, and the data object associated with the connection. Thus, the specification of a service is decoupled from the specifications of the neighboring services. Each component can have a number of attributes of primitive or user-defined data type. Data objects are represented as a tree structure comprised of the object’s attributes. Our formalism relies on the well-known “dot notation” to navigate from parent components and attributes to their children. Thus, the framework provides a uniform notation for modeling service components, connections, and complex data objects.

The problem domain is defined by an ontology of component and data types, where subtype relationships and attributes of each component and data type are described. Each type is associated with constraints that govern the permissible values of each attribute and port connection [3]. This object-oriented notation facilitates the concise representation of complex domain concepts and related integrity constraints for both service components and data types. The ontology can be formalized as constraints in the GCSP formalism. Figure 2b shows an example of a constraint that describes the *Shipper* service type (an abstract service type used in the case study described in Section 4). In the formalism, meta-variables denote placeholders for variables in a CSP representation of a (partially completed) composition problem. Here, the X may refer to a

CSP variable that represents a service component; it is constrained to a variable that has been assigned an instance of a *Shipper* service (or a subtype thereof). The antecedent of the example generic constraint further introduces local names of the CSP variables that represent the components connected to the Shipper’s *sin* and *sout* ports. The consequent of the generic constraint enforces that both connected components must represent a data connection (component type *DataConn*), whose object payload must be an instance of type *Product* (or a subtype thereof). Equality constraints between the input and output object’s *type* attribute ensures that the product (type) remains unchanged by executing the service. Furthermore, the constraint relates the component’s attributes *from* and *to* to the input and output object’s *location* attribute in order to ensure that the Shipper service only applies to products in appropriate locations (Shipper services in our example are limited to ship to limited destinations). Similar generic constraints would be created for any other type in the ontology. The resulting GCSP model allows us to concisely specify problems whose size may vary based on the initial configuration and goal (hence incorporating aspects of *dynamic CSPs* [16]) and whose structure may depend on partial value assignments (hence incorporating aspects of *conditional CSPs* [16]).

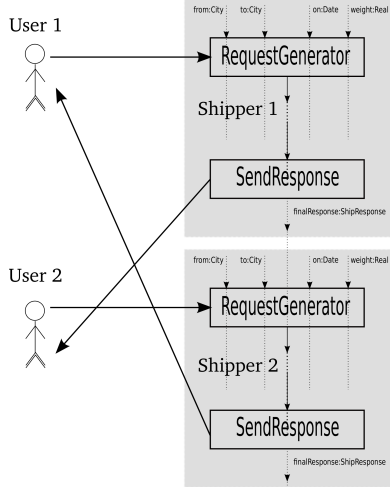
In our approach, a service composition problem is posed as a configuration task, which is expressed by an initial set of components and constraints (denoted as R_1) that must be satisfied. During the configuration process, R_n is dynamically extended by adding new variables and constraints. After each extension, R_{n+1} represents a standard CSP constraint network (without generic constraints); therefore, standard algorithms can be applied to solve the CSP. In our approach, an iterative deepening strategy that limits the number of components that may be introduced in the configuration prohibits the network from expanding indefinitely. Once that limit has been reached, the algorithm backtracks and attempts to find an alternative solutions. If that fails, the limit is increased and the search is restarted. For a detailed explanation see [12].

3 Workflow Scope

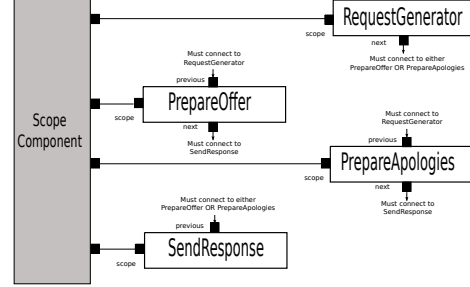
The scope of a workflow or a sub-workflow defines the tasks it encapsulates in its behavioral process specification. For example, the Shipping process from the *Producer-Shipper* composition problem [14] encapsulates tasks *RequestGenerator* and *SendResponse* to process a request and acknowledge it, respectively. Existing specifications are insufficient if multiple instances of a sub-workflow exist within a composition. For example consider the process model in Figure 1a, where two users interact with two instances of the Shipping workflow⁴. The users would like to place a shipping order using the available shipping process, but are oblivious of each other. From a composition point-of-view, components in each sub-process are interchangeable. For example, the offer requested by *User 1* to *Shipper 1* may actually be sent to *User 2* (as in Figure 1a). Hence, means to ensure messages are directed to correct recipient must be provided.

To address this problem, we introduce explicit workflow *scope components* in our generic framework. Scope components, in addition to encapsulating a process specification, also

⁴ We consider user interaction as an explicit part of the composition.



(a) Multiple Instances Problem



(b) Workflow Scope

Figure 1: Process Specifications

differentiate between multiple instances of the same workflow by maintaining a *session ID* that is unique for each occurrence of the workflow. Figure 1b shows the scope component that encapsulates the Shipping process in our example. Formally, scope components and their connections to their process elements are also defined using generic constraints like other components in our framework. Connections between scope component and the process elements are also defined as generic constraints. While each workflow instance is identified by a unique ID attribute. Additionally, each constituent component in a composition is further distinguished by its unique component ID.

4 Experimental Evaluation

We conducted an empirical investigation in order to assess the performance of our framework in different problem contexts. We investigated the time required to solve two parametric service composition problems. Our investigation were directed at assessing properties of the composition approach after potential candidates had been identified. To side-step the service matching problem, we generated a set of service instances for each service type prior to conducting our experiments. We varied the complexity of the search problem by varying the number of matching service candidates.

Our first evaluation was conducted on a generalized version of the well-known *Producer-Shipper* problem [13], where a product ordering and shipping process must be composed from individual producers and shipper services, considering the capabilities and restrictions of individual services. In this problem, a service process that consists of interactions between a producer service process and a shipper service process are to be synthesized to achieve the goal of delivering an item of particular size and cost to a given location. The producer and shipper processes themselves are composite service processes comprised of a number of tasks. Negotiation tasks between the user and both service processes must be taken into account, and the data and control flow representing the different steps and messages flowing between services must be synthesized to obtain an executable process.

Since the original problem description in [13] has only few components and can be solved almost instantaneously in our framework, we generalized the problem from a single shipping process instance to a parametrized problem that allows multiple concurrent instances. We also introduced additional requirements that ensure that only particular combinations of producer instances and shipper services may work together in a shipping process to obtain a more realistic scenario, where only few of a large number of alternative components will lead to a successful composition. These constraints link producers and shippers in different copies of the process, which can no longer be solved independently.

Our largest problem with 28 parallel producer-shipper processes (1400 services) can be solved in roughly 3 minutes. This result is quite competitive with other approaches.

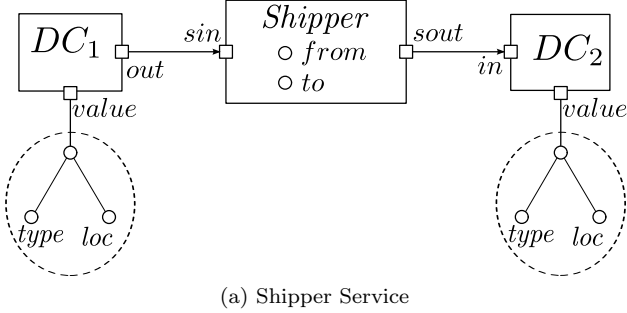
4.1 Supply Chain Coordination

The a Producer-Shipper process is quite atypical in that it does not include complex chains of services or non-trivial transformations of data exchanged between services. Hence, we conducted further experiments to assess the performance of our framework in a scenario where items must be processed by a sequence of services in order to meet the goal requirements.

Assume the supply chain of a pie factory (at a fixed location) is to be configured using services. The factory requires supplies of flour and sugar to produce pies. Our model includes wheat and sugarcane farming *services* located in various Indian cities. In addition to the factory and farmers, the problem domain includes flour and sugar mills located in a number of South-East Asian destinations. The model also includes shipping services that ship products between locations. Goods are modeled as a composite data structure containing name, quantity, and location. Each service has input and output ports that model the goods received and supplied by the service. Shipping services update the location property of goods to refer to the destination location. The aim is to compute a composition utilizing available services (farmers, mills, and shippers) in order to facilitate production operations in the pie factory while minimizing handling of the products (minimal number

of shippers).

Generic constraints permits us to model services precisely as shown in Figure 2.



(a) Shipper Service

$$\begin{aligned}
X &\sqsubseteq \text{Shipper} \wedge X.\text{sin} = DC_1 \wedge X.\text{sout} = DC_2 \Rightarrow \\
DC_1 &\sqsubseteq \text{DataConn} \wedge DC_2 \sqsubseteq \text{DataConn} \wedge \\
DC_1.\text{value} &\sqsubseteq \text{Product} \wedge DC_2.\text{value} \sqsubseteq \text{Product} \wedge \\
DC_1.\text{value.type} &= DC_2.\text{value.type} \wedge \\
DC_1.\text{value.loc} &= X.\text{from} \wedge \\
DC_2.\text{value.loc} &= X.\text{to}.
\end{aligned}$$

(b) Shipper Generic Constraint

Figure 2: Shipping Service Specification

The problem structure in our scenario requires our solver to explore different alternative matching services, and for each alternative, longer chains of prerequisite services must be devised. That extensive exploration is required at each choice point is relevant to the web service composition context, where number of (similar) services can be quite high. The following experiment aims to quantify effects of the changed search behavior on our framework.

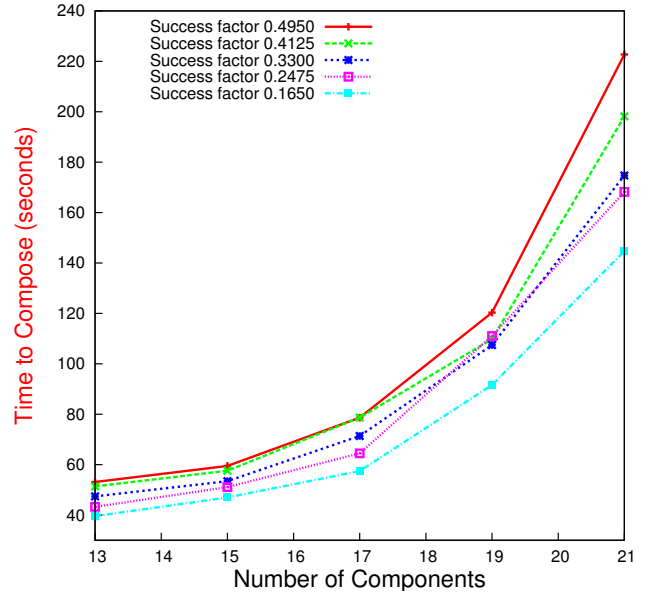
4.2 Experimental Setup and Results

We conducted experiments on a number of variants of our model. All our models contain 1 pie factory (the goal), 2 mills, and 12 farming locations. The smallest model involves choosing among 25 shipping locations. We gradually increase the number of shipping locations in each of our models; the largest model consists of 45 locations. Our smallest variant results in a composition with 13 services and our largest variant requires 21 services. The smaller model was obtained by introducing more direct shippers between service locations, while the model size was increased by removing direct shippers and introducing additional in-direct shipping routes.

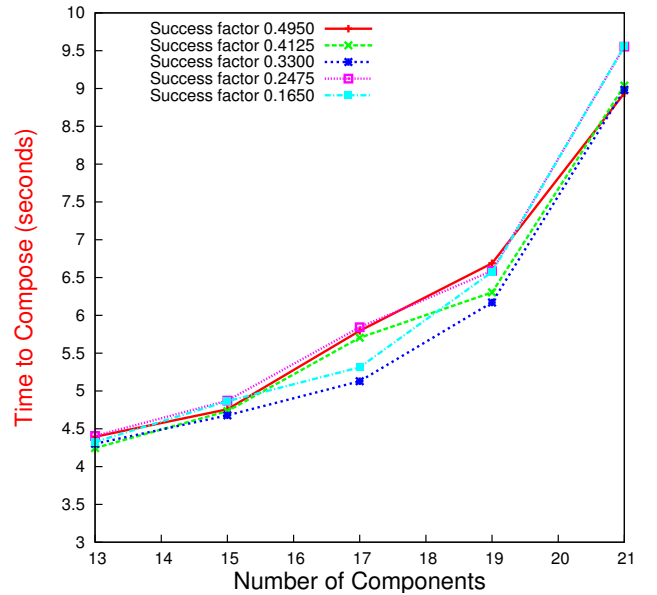
We introduce another parameter *success factor* to vary the complexity of the problem based on the availability of matching services. Consider the coordination process as a graph exploration task, where we seek to reach nodes representing farmers from the pie factory node. In such a setting, the success factor of a model is the average probability of reaching the farmer nodes through an appropriate mill node. That is, the higher the success factor is the more likely exploring an arbitrary branch will result in a successful match. A model with success factor 1 implies that exploring any out-going branch of the factory will definitely lead us to a matching services. The

Producer-Shipper scenario model has a success factor close to 1 and hence exhibits near linear performance [12]. In this experiment we only consider models with success factors below 0.5 to evaluate the performance in cases that are more likely to fail. This setting reflects the service composition assumption that many candidates that provide similar services exist, but only a few of those will be suitable. We vary the success factors of our models by replacing suitable services with ones that share similar profiles, but do not offer the capabilities required for a composition goal.

The first set of experiments employed the iterative deepening strategy discussed in [12]. The limit on the number of services in each experiment was initialized to 1 and was incremented by 1 in each iterative deepening step.



(a) Composing by Iterative Deepening



(b) Composing by Estimation

Figure 3: Pie Factory Problem Results

The results are shown in Figure 3a. Our smallest problem with 13 services can be solved in roughly 45 seconds, and our largest problem with 21 services requires roughly 3 minutes. This result is comparable in terms of problem size and complexity, and is quite competitive to other composition systems [1, 13]. Note that the time required to find a solution is depends predominantly on the size of the problem. The success factor has only minor influence.

While the approach exhibits competitive performance and supports process-level compositions (unlike [7]), the results in Figure 3a seem counter-intuitive as our results indicate that the problem instance with smallest success factor is solved fastest. Intuitively one would expect the opposite, since the chance of making a wrong choice and backtracking is more frequent in models with low success factor. To analyze this observation, we conducted another set of experiments where the number of service required in a composition is approximated beforehand (no iterative deepening). The results are presented in Figure 3b. While the graphs in Figure 3 appear similar, the scale of the ordinate is different. It can be observed that the time taken to solve models with high success factor is lesser than that of model with low success factor. Therefore, backtracking caused by the iterative deepening strategy is the cause for the counter-intuitive observation in Figure 3a.

5 Conclusion

We outlined basic elements of our composition framework and showed that our formalism can represent component types, data types, and data flow constraints require in complex service composition scenarios. We presented process-specific constraints that enforce control and data flow properties within a restricted sub-scope of a larger process. We outlined an iterative approach to translate a composition problem formalized as Generic CSP problem into a series of classical CSP problems, and showed the practicality of our framework on two case studies. We obtained that our framework is competitive with other published service composition approaches and showed that our approach is robust with respect to the sparsity of solutions. We also showed that significant performance improvements can be achieved if additional information about the size of the solution is incorporated into the solving process.

We are currently exploring strategies to further improve scalability by avoiding to undo assignments in complete (successful) branches of the search space. Particularly, we are investigating the effect of replacing backtracking with a back jumping strategy that arbitrarily chooses a previous assignment to revise. Our current solving strategy is predominantly concerned with finding a *valid* solution comprised of a small number of components. Inferring the *optimal* solution for a configuration goal has played a subordinate role in our framework, since configurations comprising few components are often satisficing. Mechanisms to incorporate further guidance heuristics and optimization strategies into our process configuration framework are possible avenues for future research.

REFERENCES

- [1] Patrick Albert, Laurent Henocque, and Mathias Kleiner, ‘Configuration Based Workflow Composition’, in *Proc. of ICWS 2005*, (2005).

- [2] Matthias Born, Jörg Hoffmann, Tomasz Kaczmarek, Marek Kowalkiewicz, Ivan Markovic, James Scicluna, Ingo Weber, and Xuan Zhou, ‘Semantic Annotation and Composition of Business Processes with Maestro’, in *Proceedings of the European Semantic Web Conference (ESWC)*, (2008).
- [3] Gerhard Fleischanderl et al., ‘Configuring large-scale systems with generative constraint satisfaction’, *IEEE Intelligent Systems*, **13**(4), (1998).
- [4] Alois Haselböck and Markus Stumptner, ‘An integrated approach for modelling complex configuration domains’, in *Proc. of the 13th Int. Conf. on Expert Systems, AI, and Natural Language*, (1993).
- [5] Ahlem Ben Hassine, Shigeo Matsubara, and Toru Ishida, ‘A Constraint-Based Approach to Horizontal Web Service Composition’, in *Proc. ISWC*, (2006).
- [6] E. Karakoc and P. Senkul, ‘Composing semantic web services under constraints’, *Expert Syst. Appl.*, **36**(8), 11021–11029, (2009).
- [7] Freddy Lécué, Alexandre Delteil, and Alain Léger, ‘Optimizing Causal Link Based Web Service Composition’, in *ECAI*, (2008).
- [8] Lei Li and Ian Horrocks, ‘A software framework for matchmaking based on Semantic Web technology’, in *Proc. of WWW*, pp. 331–339, Budapest, (2003).
- [9] Zhen Liu, Anand Ranganathan, and Anton Riabov, ‘A planning-based approach for the automated configuration of the enterprise service bus’, in *Proc. ICSOC*, (2008).
- [10] Adriana Lopez and Fahiem Bacchus, ‘Generalizing GraphPlan by Formulating Planning as a CSP’, in *Proc. IJCAI*, (2003).
- [11] Annapaola Marconi, Marco Pistore, Piero Poccianti, and Paolo Traverso, ‘Automated Web Service Composition at Work: the Amazon/MPS Case Study’, in *Proc. of ICWS*, (2007).
- [12] Wolfgang Mayer, Rajesh Thiagarajan, and Markus Stumptner, ‘Service Composition As Generative Constraint Satisfaction’, in *Proc. of ICWS*, (2009).
- [13] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso, ‘Automated composition of web services by planning at the knowledge level’, in *Proc. IJCAI*, (2005).
- [14] Marco Pistore, Paolo Traverso, Piergiorgio Bertoli, and A. Marconi, ‘Automated synthesis of composite BPEL4WS web services’, in *ICWS*, pp. 293–301, Orlando, (2005).
- [15] Cédric Pralet and Gérard Verfaillie, ‘Using constraint networks on timelines to model and solve planning and scheduling problems’, in *Proc. ICAPS*, (2008).
- [16] *Handbook of Constraint Programming*, eds., F. Rossi, P. van Beek, and T. Walsh, Elsevier, 2006.
- [17] Ignacio Silva-Lepe, Revathi Subramanian, Isabelle Rouvelou, Thomas A. Mikalsen, Judah Diamant, and Arun Iyengar, ‘Soalive service catalog: A simplified approach to describing, discovering and composing situational enterprise services’, in *Proc. ICSOC*, (2008).
- [18] Rajesh Thiagarajan and Markus Stumptner, ‘Service Composition With Consistency-based Matchmaking: A CSP-based Approach’, in *Proc. ECOWS*, (2007).
- [19] Markus Zanker, Dietmar Jannach, Marius-Calin Silaghi, and Gerhard Friedrich, ‘A distributed generative csp framework for multi-site product configuration’, in *12th International Workshop on Cooperative Information Agents (CIA)*, (2008).

Encoding the Linux Kernel Configuration in Propositional Logic

Christoph Zengler and Wolfgang Kuchlin¹

Abstract. We present a formalization of the Linux Kernel configuration and propose a set of rules how to encode this formalization in propositional logic. The resulting propositional formulas describe all valid configurations of the Linux Kernel with respect to one specific hardware architecture. The advantage of a formula in propositional logic is that we can use all the elaborate tools and techniques from the SAT community like SAT solvers, parametric SAT solvers, or model counters. We show how we can use these available tools to perform e.g. an automated search for redundant or necessary options or automatically produce configuration variants. We have implemented our approach and compiled the formulas for all available hardware architectures. Based on this implementation, we show some experimental results on size and complexity of the resulting formulas.

1 INTRODUCTION

The monolithic Linux Kernel is a highly configurable software system. The Linux Kernel supports more than 60 different hardware platforms grouped in about 25 hardware architectures. There are various kernel sub-systems e.g. for memory management, file systems, or network protocols. There are thousands of device drivers for all different kinds of hardware (e.g. scanners, ISDN cards, video cards, etc.) [1]. When compiling the kernel one can choose between thousands of options. These options are organized in the so called `KConfig` files for the `kbuild` system. There are over 1000 `KConfig` files, which – combined together – describe the complete Linux Kernel Configuration (LKC).

Our approach is to translate these files into one propositional formula for each hardware architecture. This formula then describes all valid configurations. A formalization of configuration problems in propositional logic is a well studied approach which is suitable for large scale industrial problems [6]. We show how to use standard methods and tools for propositional formulas to test, manipulate and list kernel configurations.

In [4] the LKC is described in terms of a software product line. In [5] the same authors present a Linux Kernel configuration tool. Post and Sinz describe a technique called *Configuration Lifting* [3]. Their idea is to directly encode certain options into a program that should be verified. Thus you cannot only verify that a program is correct for one specific configuration, but for all possible configurations (wrt. to the lifted configuration options).

To the best of our knowledge our approach presented in this paper is new and merits further research. None of the above mentioned approaches compiles the complete LKC into one formula, which can then be verified. With our methods we can solve problems that could

to the best of our knowledge not be solved before e.g. automatically produce variants wrt. a dedicated set of options or counting the number of valid configurations. Also, our approach is not only for users, configuring the kernel, but also for developers, checking the validity of their documentation of the options. All described methods are implemented and we present some experimental results in Section 5.

2 THE LINUX KERNEL CONFIGURATION

For most options the user can choose between three states: (`y`) The respective option is compiled directly into the kernel; (`m`) the option is compiled as module and can be loaded and unloaded into the kernel at runtime; (`n`) the option is not available at all. Options can depend on the state of other options (*dependencies*) or can influence the state of other options (*selections*). All options, their dependencies, and their selections are described in the `KConfig` files.

2.1 Configuration Options

A single `KConfig` file is a collection of configuration options (also referred to as configuration symbols). There are five different types t of configuration options: `tristate`, `bool`, `string`, `int`, and `hex`. We denote their respective sets by $\mathcal{O}_T$, $\mathcal{O}_B$, $\mathcal{O}_S$, $\mathcal{O}_I$, and $\mathcal{O}_H$. The set of all configuration options is denoted by $\mathcal{O} = \mathcal{O}_T \cup \mathcal{O}_B \cup \mathcal{O}_S \cup \mathcal{O}_I \cup \mathcal{O}_H$. We use uppercase letters ‘ A ’, ‘ B ’, ... to denote configuration options of $\mathcal{O}$. Each set S of options has its own domain $\text{dom}(S)$. The domains for a set expand in a natural way to elements of this set such that for each option $s \in S$ with $\text{dom}(S) = D$ we have $\text{dom}(s) = D$. The domains for the five different sets of configuration options are: $\text{dom}(\mathcal{O}_T) = \{0, 1, 2\}$, $\text{dom}(\mathcal{O}_B) = \{0, 2\}$, $\text{dom}(\mathcal{O}_S) = \mathbb{S}$, $\text{dom}(\mathcal{O}_I) = \mathbb{S}_I$, $\text{dom}(\mathcal{O}_H) = \mathbb{S}_H$. $\mathbb{S}$ is the set of all valid strings, $\mathbb{S}_I$ is the set of all valid strings representing an integer, $\mathbb{S}_H$ is the set of all valid strings representing a number in hexadecimal format. We can clearly see, that $\text{dom}(\mathcal{O}_B) \subset \text{dom}(\mathcal{O}_T)$, $\text{dom}(\mathcal{O}_I) \subset \text{dom}(\mathcal{O}_S)$, and $\text{dom}(\mathcal{O}_H) \subset \text{dom}(\mathcal{O}_S)$. Thus all five types can be derived from the two main types `tristate` and `string`.

Since there are three different states for all `tristate` symbols $s \in \mathcal{O}_T$ and `bool` symbols can be derived from `tristate` symbols, the configuration files use a tristate logic for expressions over configuration options of $\mathcal{O}_T \cup \mathcal{O}_B$. This logic uses one more state than normal boolean logic to express the module state. For calculations we identify `n` with 0, `m` with 1, and `y` with 2. An *option assignment* is a partial function $\alpha_{\mathcal{O}} : \mathcal{O} \rightarrow \{0, 1, 2\} \cup \mathbb{S}$ mapping configuration options of $\mathcal{O}_T \cup \mathcal{O}_B$ to one of the three states 0, 1, 2 and options of $\mathcal{O}_S \cup \mathcal{O}_I \cup \mathcal{O}_H$ to their corresponding string value. We can now extend the definitions of ‘not’ (!), ‘and’ (&&), and ‘or’ (||) in

¹ Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, email: [zengler,kuechlin]@informatik.uni-tuebingen.de

a natural way. For an option $A \in \mathcal{O}$ and arbitrary tristate expression $e_0, \dots, e_n$, we define the evaluation β of a tristate expression:

$$\begin{aligned}\beta(\mathbf{y}) &= 2 \\ \beta(\mathbf{m}) &= 1 \\ \beta(\mathbf{n}) &= 0 \\ \beta(A) &= \alpha_{\mathcal{O}}(A) \\ \beta(!e_0) &= 2 - \beta(e_0) \\ \beta(e_0 \& \dots \& e_n) &= \min(\beta(e_0), \dots, \beta(e_n)) \\ \beta(e_0 || \dots || e_n) &= \max(\beta(e_0), \dots, \beta(e_n))\end{aligned}$$

An expression symbol s^e is either a regular configuration symbol $s \in \mathcal{O}$ or an arbitrary string $\mathfrak{s} \in \mathbb{S}$. Expression symbols can be compared with '=' and '!=', which are valid expressions. We extend β with:

$$\begin{aligned}\beta(s_1^e = s_2^e) &= 2 \text{ if } \beta(s_1^e) = \beta(s_2^e), 0 \text{ else} \\ \beta(s_1^e \neq s_2^e) &= 0 \text{ if } \beta(s_1^e) = \beta(s_2^e), 2 \text{ else}\end{aligned}$$

Remark. In fact, the LKC files contain only (dis)equalities of the form $A = B$ and $X = \mathfrak{s}$ with $A, B \in \mathcal{O}_T \cup \mathcal{O}_B$ and $X \in \mathcal{O}_S \cup \mathcal{O}_H \cup \mathcal{O}_I$. For the rest of the paper we will only consider equalities of this form.

2.2 Configuration Files

There are about 1000 files describing the configuration database of the LKC. These files can be merged into one big file in a pre-processing step. We will present only the parts of the grammar of these files that are important for our approach. For this initial formalization we omit structures like help texts, comments, or default values.

Table 1 summarizes the abstract syntax. $s \in \mathcal{O}$ is a configuration option, $\mathfrak{s} \in \mathbb{S}$ is an arbitrary string. We state meta symbols for each element. The meta symbols can include the parameters of the element e.g. in $b^I(E, \vec{B})$. If the parameters are not required, we use just the symbol without parameters, thus b^I .

A single configuration file $f(\vec{b})$ consists of a set of blocks $\vec{b}$. The most important block type is the configuration block $b^C(s, a^T, \vec{a}^D, \vec{a}^S)$, which describes a single configuration option $s \in \mathcal{O}$. An option s must have a type attribute a^T , which fixes the type of s to one of the five possibilities discussed above. s can have an arbitrary number of dependence and selection attributes.

A dependence attribute $a^D(e)$ describes a tristate expression e on which the current option depends. Thus e fixes an upper bound for s , meaning if $\beta(e) = 0$, then $\alpha_{\mathcal{O}}(s)$ must be 0 too, if $\beta(e) = 1$, then $\alpha_{\mathcal{O}}(s)$ can be 0 or 1, and if $\beta(e) = 2$, then s can be assigned to any state of $\{0, 1, 2\}$.

A selection attribute $a^S(s', e)$ specifies a symbol s' which must at least be assigned to the value of the current option s . Thus s fixes a lower bound for s' , meaning if $\alpha_{\mathcal{O}}(s) = 2$, s' must be assigned to 2 too, if $\alpha_{\mathcal{O}}(s) = 1$, s' can be assigned to 1 or 2, and if $\alpha_{\mathcal{O}}(s) = 0$ then s' can be assigned to any state of $\{0, 1, 2\}$. A selection attribute can have an additional tristate expression e as guarding condition. The selection will only be performed if $\beta(e) = 2$.

The menu, conditional, and choice blocks help to organize the configuration options. A menu block $b^M(\mathfrak{s}, \vec{a}^D, \vec{b})$ groups several other blocks $\vec{b}$. The dependencies $\vec{a}^D$ are propagated to all children blocks $\vec{b}$. Thus

```
menu "some text"
  depends on A

  config B
    type tristate

  config C
    type bool
endmenu

can be replaced by

config B
  type tristate
  depends on A

config C
  type bool
  depends on A
```

A conditional block $b^I(e, \vec{b})$ also groups several other blocks $\vec{b}$ but adds only one dependence e to each of them. A choice block $b^H(t, \vec{a}^D, \vec{b}^C)$ groups several configuration options $\vec{b}^C$. A choice block must have a type t . If t is `bool` it means, that at most one of the enclosed options can be assigned to 2, all other options must be assigned to 0. In a `tristate` choice, at most one option can be set to 2, but an arbitrary number of options can be assigned to 1. If a choice block has dependencies $\vec{a}^D$, they are propagated to each child configuration option of $\vec{b}^C$.

3 TRANSLATION INTO PROPOSITIONAL LOGIC

For the propositional logic encoding of the configuration files, we propose a two step procedure. In the first step we build a symbol database containing all configuration symbols with their respective types, dependencies, and selections and a choice database containing all choices with their respective types and options. In the second step we translate these two databases into propositional logic.

We use the standard notation of propositional logic with propositional variables from a suitable infinite set $\mathcal{V}$, Boolean operators $\neg$, $\vee$, $\wedge$, $\rightarrow$, and $\leftrightarrow$, and Boolean constants 1 and 0. An *assignment* (in contrast to an option assignment) is a partial function $\alpha : \mathcal{V} \rightarrow \{0, 1\}$ mapping variables to truth values. We write $x \leftarrow b$ for $\alpha(x) = b$.

3.1 Step 1: Creating the Symbol and Choice Database

The symbol database $\mathcal{D}_S$ is a set of pairs (s, desc) , where $\text{desc} = (t, \vec{e}_1, (\vec{s}', e_2))$ is a symbol descriptor consisting of the type t of s , the set of all (also the propagated) dependencies $\vec{e}_1$ of s and the set of all selections $(\vec{s}', e_2)$ of s . A single selection (s', e_2) contains the symbol s' that is selected from the current symbol s and the condition e_2 which guards this selection. $e_2 = \mathbf{y}$ if there is no guarding condition for this selection.

For the creation of the symbol database we propagate all dependencies of the menu, conditional and choice blocks as described in Subsection 2.2. After this step we have only configuration blocks left, which can then be translated straight forward into the corresponding symbol descriptors. The function `createDBS()` in Figure 1

Table 1. Abstract syntax for the LKC files

Level	Element	Abstract Syntax		Meta Symbol
Root level	Configuration file	F	$::= \vec{B}$	$f(\vec{B})$
Block level	block	B	$::= B_C \mid B_H \mid B_M \mid B_I$	b
	configuration	B_C	$::= \text{config } s \ A_T \ \vec{A_D} \ \vec{A_S}$	$b^C(s, A_T, \vec{A_D}, \vec{A_S})$
	menu	B_M	$::= \text{menu } s \ \vec{A_D} \ \vec{B} \ \text{endmenu}$	$b^M(s, \vec{A_D}, \vec{B})$
	conditional	B_I	$::= \text{if } E \ \vec{B} \ \text{endif}$	$b^I(E, \vec{B})$
	choice	B_H	$::= \text{choice } (\text{tristate} \mid \text{bool}) \ \vec{A_D} \ \vec{B_C} \ \text{endchoice}$	$b^H(t, \vec{A_D}, \vec{B_C})$
Attribute level	attribute	A	$::= A_T \mid A_D \mid A_S \mid A_R$	a
	type	A_T	$::= \text{tristate} \mid \text{bool} \mid \text{string} \mid \text{int} \mid \text{hex}$	a^T
	dependence	A_D	$::= \text{depends on } E$	$a^D(E)$
	selection	A_S	$::= \text{select } s \ [\text{if } E]$	$a^S(s, E)$
Expression level	expression	E	$::= S_E \mid (E) \mid !E \mid E \ \&\& \ E \mid E \ \mid \mid \ E \mid$ $S_E = S_E \mid S_E \ != \ S_E \mid y \mid n \mid m$	e
	exp-symbol	S_E	$::= s \mid s$	s^e

states the exact procedure. For a file $f(\vec{b})$ it is initially called with $\text{createDBs}(\vec{b}, \emptyset)$.

Example 1 Consider the following file $f(\vec{b})$:

```

config X
  string

config B
  tristate

if B
  config C
    tristate
endif

```

```

config A
  tristate
  depends on !B
  select C if X = "s"

```

The method $\text{createDBs}(\vec{b}, \emptyset)$ yields the following symbol database:

$$\mathcal{D}_S = \{(X, (\text{string}, \emptyset, \emptyset)), (B, (\text{tristate}, \emptyset, \emptyset)), (C, (\text{tristate}, \{B\}, \emptyset)), (A, (\text{tristate}, \{!B\}, \{(C, X = "s")\}))\}$$

The choice database $\mathcal{D}_C$ is a set of pairs $(t, \vec{s})$, where t is the type of the respective choice and $\vec{s}$ is the set of configuration symbols in the respective choice. For the creation of the choice database only choice blocks of the form

$$b^H\left(t, \vec{a_D}, \left\{b^C\left(s_1, a_1^T, \vec{a_D}, \vec{a_S}\right), \dots, b^C\left(s_n, a_n^T, \vec{a_D}, \vec{a_S}\right)\right\}\right)$$

have to be considered. This block is then translated into the pair $(t, \{s_1, \dots, s_n\})$.

Example 2 The choice database for the block

```

choice
  tristate

  config A
    tristate

  config B
    tristate

  config C
    bool
endchoice

```

is $\mathcal{D}_C = \{(\text{tristate}, \{A, B, C\})\}$.

3.2 Step 2: Translating the Databases

In the second step $\mathcal{D}_S$ and $\mathcal{D}_C$ are translated into corresponding encodings in propositional logic. Since dependencies and selections are represented as tristate expressions e in the symbol descriptor **desc**, we present a way how to encode the tristate logic of these expressions into a corresponding propositional representation. Afterwards we show how to encode the dependence and selection sets, before finally translating the choice database. At the end of the translation process we will have four sets: The set $\mathcal{C}_O$ of constraints for the different configuration option types, the set $\mathcal{C}_D$ of dependency constraints, the set $\mathcal{C}_S$ of selection constraints, and the set $\mathcal{C}_C$ of choice constraints. A complete encoding φ , describing all valid configurations of the Linux kernel is then given by:

$$\varphi = \bigwedge_{t \in \mathcal{C}_O} t \wedge \bigwedge_{d \in \mathcal{C}_D} d \wedge \bigwedge_{s \in \mathcal{C}_S} s \wedge \bigwedge_{c \in \mathcal{C}_C} c$$

3.2.1 Translation of the Configuration Symbols

As mentioned in Subsection 2.1, all configuration symbol types can be derived from **tristate** and **string**. We encode the three

$$\begin{aligned}
\text{createDB}_S(\{b_1, \dots, b_n\}, \vec{e}) &= \text{createDB}_S(\{b_1\}, \vec{e}) \cup \dots \cup \text{createDB}_S(\{b_n\}, \vec{e}) \\
\text{createDB}_S\left(\left\{b^C\left(s, a^T, \vec{a}^D, \vec{a}^S\right)\right\}, \vec{e}\right) &= \left\{\left(s, \text{analyzeSymb}\left(a^T, \vec{a}^D, \vec{a}^S, \vec{e}\right)\right)\right\} \\
\text{createDB}_S\left(\left\{b^M\left(\mathfrak{s}, \vec{a}^D, \vec{b}\right)\right\}, \vec{e}\right) &= \text{createDB}_S\left(\vec{b}, \vec{e} \cup \text{deps}\left(\vec{a}^D\right)\right) \\
\text{createDB}_S\left(\left\{b^I\left(e_1, \vec{b}\right)\right\}, \vec{e}\right) &= \text{createDB}_S\left(\vec{b}, \vec{e} \cup \{e_1\}\right) \\
\text{createDB}_S\left(\left\{b^H\left(a^T, \vec{a}^D, \vec{b}^C\right)\right\}, \vec{e}\right) &= \text{createDB}_S\left(\vec{b}^C, \vec{e} \cup \text{deps}\left(\vec{a}^D\right)\right) \\
\text{analyzeSymb}\left(a^T, \vec{a}^D, \vec{a}^S, \vec{e}\right) &= \left(a^T, \text{deps}\left(\vec{a}^D\right) \cup \vec{e}, \text{sels}\left(\vec{a}^S\right)\right) \\
\text{deps}\left(\left\{a^D\left(e_1\right), \dots, a^D\left(e_n\right)\right\}\right) &= \{e_1, \dots, e_n\} \\
\text{sels}\left(\left\{a^S\left(s_1, e_1\right), \dots, a^S\left(s_n, e_n\right)\right\}\right) &= \{(s_1, e_1), \dots, (s_n, e_n)\}
\end{aligned}$$

Figure 1. The function $\text{createDB}_S()$ for creating the symbol database

different states of the tristate logic with two bits in propositional logic. Each symbol $A \in \mathcal{O}_T \cup \mathcal{O}_B$ is translated into two variables $a_0, a_1 \in \mathcal{V}$, where $\mathcal{V}$ is the set of propositional variables of our propositional encoding. The assignment $\{a_0 \leftarrow 0, a_1 \leftarrow 0\}$ represents $\alpha_{\mathcal{O}}(A) = 0$, $\{a_0 \leftarrow 0, a_1 \leftarrow 1\}$ represents $\alpha_{\mathcal{O}}(A) = 1$, and $\{a_0 \leftarrow 1, a_1 \leftarrow 0\}$ represents $\alpha_{\mathcal{O}}(A) = 2$. Since the assignment $\{a_0 \leftarrow 1, a_1 \leftarrow 1\}$ is not possible, for each $A \in \mathcal{O}_T$ the constraint $(\neg a_0 \vee \neg a_1)$ is added to $\mathcal{C}_O$. For each symbol $A \in \mathcal{O}_B$ we add the constraint $\neg a_1$ to $\mathcal{C}_O$ in order to exclude the module state $\{a_0 \leftarrow 0, a_1 \leftarrow 1\}$ (and implicitly also the illegal assignment $\{a_0 \leftarrow 1, a_1 \leftarrow 1\}$).

Remark. In fact one could translate each variable $A \in \mathcal{O}_B$ with only one propositional variable $a \in \mathcal{V}$. For a uniform presentation we chose to treat *bool* and *tristate* variables the same. However, since we add the constraint $\neg a_1$ to $\mathcal{C}_O$ for each $A \in \mathcal{O}_B$, all superfluous variables can be easily eliminated by unit propagation.

As seen in Subsection 2.1, the only valid tristate expressions containing symbols of the type *string*, *int*, or *hex* are equalities and disequalities. For a configuration symbol $X \in \mathcal{O}_S \cup \mathcal{O}_I \cup \mathcal{O}_H$ we collect all strings $\mathfrak{s}$ occurring on the right side of an equality or disequality in a set $\text{eq}(X)$.

$$\text{eq}(X) = \{\mathfrak{s} \in \mathbb{S} \mid \text{there exists an expression } X = \mathfrak{s} \text{ or } X \neq \mathfrak{s}\}$$

We introduce new variables $x_{\mathfrak{s}} \in \mathcal{V}$, for each $\mathfrak{s} \in \text{eq}(X)$. The assignment $x_{\mathfrak{s}} \leftarrow 1$ represents the option assignment $\alpha_{\mathcal{O}}(X) = \mathfrak{s}$. Since each configuration symbol $X \in \mathcal{O}_S \cup \mathcal{O}_I \cup \mathcal{O}_H$ must be assigned to exactly one option of $\text{eq}(X)$, we add for each symbol $X \in \mathcal{O}_S \cup \mathcal{O}_I \cup \mathcal{O}_H$ the following constraint to $\mathcal{C}_O$:

$$\left(\bigvee_{\mathfrak{s} \in \text{eq}(X)} x_{\mathfrak{s}}\right) \wedge \bigwedge_{\mathfrak{s} \in \text{eq}(X)} \left(x_{\mathfrak{s}} \rightarrow \bigwedge_{\mathfrak{s}' \in \text{eq}(X) \setminus \{\mathfrak{s}\}} \neg x_{\mathfrak{s}'}\right)$$

3.2.2 Translation of the Symbol Database

For each tristate expression e we define two projections $\pi_0(e)$ and $\pi_1(e)$, representing the first and second bit of this expression's propositional encoding. For configuration symbols $A, B \in$

$\mathcal{O}_T \cup \mathcal{O}_B$, $X \in \mathcal{O}_S \cup \mathcal{O}_I \cup \mathcal{O}_H$ and arbitrary tristate expressions $e_0, \dots, e_n$, expression symbols s_1^e, s_2^e , and strings $\mathfrak{s}$, the definitions of π_0 and π_1 are stated in Table 2.

Example 3 Consider a tristate expression $e = A \&\& !B$. Then $\pi_0(e) = a_0 \wedge \neg b_0 \wedge \neg b_1$ and $\pi_1(e) = (a_0 \vee a_1) \wedge ((\neg b_0 \wedge \neg b_1) \vee b_1) \wedge (a_1 \vee b_1)$. We present the truth table for all possible values of A and B . One can clearly see the correspondence between $\beta(e)$ and the translations $\pi_0(e)$ and $\pi_1(e)$.

A	B	a_0	a_1	b_0	b_1	$\beta(e)$	$\pi_0(e)$	$\pi_1(e)$
0	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0
0	2	0	0	1	0	0	0	0
1	0	0	1	0	0	1	0	1
1	1	0	1	0	1	1	0	1
1	2	0	1	1	0	0	0	0
2	0	1	0	0	0	2	1	0
2	1	1	0	0	1	1	0	1
2	2	1	0	1	0	0	0	0

For each pair $(s, \text{desc}) \in \mathcal{D}_S$ we add dependency and selection constraints to $\mathcal{C}_D$ and $\mathcal{C}_S$. We have already seen in Subsection 2.2 how dependencies fix an upper bound for the current symbol. The following translation $\pi_D(s, e)$ of a dependence e for a symbol s assures that $\alpha_{\mathcal{O}}(s) \leq \beta(e)$:

$$\pi_D(s, e) = \overbrace{(\neg \pi_0(e) \wedge \neg \pi_1(e) \rightarrow \neg s_0 \wedge \neg s_1)}^{\beta(e)=0 \rightarrow \alpha_{\mathcal{O}}(s)=0} \wedge \overbrace{(\pi_1(e) \rightarrow \neg s_0)}^{\beta(e)=1 \rightarrow \alpha_{\mathcal{O}}(s) \neq 2}$$

The selection of a symbol s' for an option s under condition e fixes a lower bound for s' . The translation $\pi_S(s, s', e)$ assures, that (given that e holds) $\alpha_{\mathcal{O}}(s') \geq \alpha_{\mathcal{O}}(s)$:

$$\pi_S(s, s', e) = \overbrace{(s_0 \wedge \pi_0(e) \rightarrow s'_0)}^{\alpha_{\mathcal{O}}(s)=2 \rightarrow \alpha_{\mathcal{O}}(s')=2} \wedge \overbrace{(s_1 \wedge \pi_0(e) \rightarrow s'_0 \vee s'_1)}^{\alpha_{\mathcal{O}}(s)=1 \rightarrow \alpha_{\mathcal{O}}(s') \neq 0}$$

For each entry $(s, \text{desc}) \in \mathcal{D}_S$ with

$$\text{desc} = (t, \{e_1, \dots, e_n\}, \{(s'_1, e'_1), \dots, (s'_m, e'_m)\})$$

Table 2. Translation rules for tristate expressions e'

e'	$\pi_0(e')$	$\pi_1(e')$
A	a_0	a_1
$!e$	$\neg\pi_0(e) \wedge \neg\pi_1(e)$	$\pi_1(e)$
$e_0 \& \dots \& e_n$	$\pi_0(e_0) \wedge \dots \wedge \pi_0(e_n)$	$\bigwedge_{i \in \{0, \dots, n\}} (\pi_0(e_i) \vee \pi_1(e_i)) \wedge \bigvee_{i \in \{0, \dots, n\}} \pi_1(e_i)$
$e_0 \dots e_n$	$\pi_0(e_0) \vee \dots \vee \pi_0(e_n)$	$\bigwedge_{i \in \{0, \dots, n\}} (\neg\pi_0(e_i)) \wedge \bigvee_{i \in \{0, \dots, n\}} \pi_1(e_i)$
$A=B$	$a_0 \leftrightarrow b_0 \wedge a_1 \leftrightarrow b_1$	0
$X=s$	x_s	0
$s_1^e != s_2^e$	$\neg\pi_0(s_1^e = s_2^e)$	0

we add the constraints $\pi_D(s, e_1), \dots, \pi_D(s, e_n)$ to $\mathcal{C}_D$ and we add the constraints $\pi_S(s, s'_1, e'_1), \dots, \pi_S(s, s'_m, e'_m)$ to $\mathcal{C}_S$.

Example 4 Reconsider the blocks from Example 1. We constructed the symbol database

$$\begin{aligned} \mathcal{D}_S = & \{(X, (string, \emptyset, \emptyset)), \\ & (B, (tristate, \emptyset, \emptyset)), \\ & (C, (tristate, \{B\}, \emptyset)), \\ & (A, (tristate, \{!B\}, \{(C, X = "s")\}))\} \end{aligned}$$

For this database we have

$$\begin{aligned} \mathcal{C}_O &= \{x_s, \neg b_0 \vee \neg b_1, \neg c_0 \vee \neg c_1, \neg a_0 \vee \neg a_1\} \\ \mathcal{C}_D &= \{\pi_D(C, B), \pi_D(A, !B)\} \\ \mathcal{C}_S &= \{\pi_S(A, C, X = "s")\} \end{aligned}$$

with

$$\begin{aligned} \pi_D(C, B) &= ((\neg b_0 \wedge \neg b_1) \rightarrow (\neg c_0 \wedge \neg c_1)) \wedge (b_1 \rightarrow \neg c_0) \\ \pi_D(A, !B) &= (b_0 \rightarrow (\neg a_0 \wedge \neg a_1)) \wedge (b_1 \rightarrow \neg a_0) \end{aligned}$$

and

$$\pi_S(A, C, X = "s") = (a_0 \wedge x_s \rightarrow c_0) \wedge (a_1 \wedge x_s \rightarrow c_0 \vee c_1)$$

3.2.3 Translation of the Choice Database

In Subsection 2.2 we have seen, that there are only two types for choices: `tristate` and `bool`. In a `bool` choice, only one of the enclosed options can be set to 2, all other options must be set to 0 (the module state 1 is not allowed at all). For each `bool` choice $(\text{bool}, \{s_1, \dots, s_n\}) \in \mathcal{D}_C$ we add the constraint

$$\bigwedge_{i \in \{1, \dots, n\}} \neg\pi_1(s_i) \wedge \bigwedge_{i \in \{1, \dots, n\}} \left(\pi_0(s_i) \rightarrow \bigwedge_{j \in \{1, \dots, n\} \setminus \{i\}} \neg\pi_0(s_j) \right)$$

to $\mathcal{C}_C$. In a `tristate` choice also only one option can be set to 2, but an arbitrary number of options can be set to 1 additionally. For each choice $(\text{tristate}, \{s_1, \dots, s_n\}) \in \mathcal{D}_C$ we add the constraint

$$\bigwedge_{i \in \{1, \dots, n\}} \left(\pi_0(s_i) \rightarrow \bigwedge_{j \in \{1, \dots, n\} \setminus \{i\}} \neg\pi_0(s_j) \right)$$

to $\mathcal{C}_C$.

Example 5 For the `tristate` choice of Example 2 we add the following constraint C to $\mathcal{C}_C$:

$$C = (a_0 \rightarrow \neg b_0 \wedge \neg c_0) \wedge (b_0 \rightarrow \neg a_0 \wedge \neg c_0) \wedge (c_0 \rightarrow \neg a_0 \wedge \neg b_0)$$

If the choice would be of type `bool`, we would add

$$\neg a_1 \wedge \neg b_1 \wedge \neg c_1 \wedge C.$$

4 POSSIBLE APPLICATIONS

The formula $\varphi = \bigwedge_{t \in \mathcal{C}_O} t \wedge \bigwedge_{d \in \mathcal{C}_D} d \wedge \bigwedge_{s \in \mathcal{C}_S} s \wedge \bigwedge_{c \in \mathcal{C}_C} c$ constructed in the last section describes all valid Linux kernel configurations in a single propositional formula. Now we can use all the techniques from the SAT area to process this formula. We will outline two possible fields of application of these techniques: (1) automatic detection of redundant or necessary options (2) automatic listing of all configuration variants.

4.1 Detecting redundant and necessary options

Obviously the formula φ should be satisfiable otherwise there would be no valid Linux Kernel configuration. A state-of-the-art SAT solver can compute this in < 0.1 s. However, the interesting question is if the formula is still satisfiable when some options are fixed. E.g. for $A \in \mathcal{O}_T$ one can check if $\varphi \wedge a_0$ is satisfiable. If not, then the option A can never be compiled directly into the kernel. If $\varphi \wedge a_1$ is not satisfiable, option A cannot be compiled as module. If both $\varphi \wedge a_0$ and $\varphi \wedge a_1$ are not satisfiable, we know that option A cannot be chosen in any valid configuration of the kernel. We have then detected a *redundant* option.

If $\varphi \wedge a_1$ and $\varphi \wedge \neg a_0 \wedge \neg a_1$ are both not satisfiable, we know that option A must be set to 2 in any valid configuration. We then refer to A as *necessary* option. Of course one can build more complex extensions of the formula φ for checking a specific choice of options. One can also automate these tests and search the configuration files systematically for redundant or necessary options.

Example 6 Reconsider the blocks from Example 1 with the sets $\mathcal{C}_O$, $\mathcal{C}_D$ and $\mathcal{C}_S$ given in Example 4. If we want to compile A directly into the kernel, we have to check the formula

$$\bigwedge_{t \in \mathcal{C}_O} t \wedge \bigwedge_{d \in \mathcal{C}_D} d \wedge \bigwedge_{s \in \mathcal{C}_S} s \wedge a_0$$

with a SAT solver. We get the result “false”, meaning we cannot set A to 2 in any valid configuration. If we check the formula

$$\bigwedge_{t \in \mathcal{C}_O} t \wedge \bigwedge_{d \in \mathcal{C}_D} d \wedge \bigwedge_{s \in \mathcal{C}_S} s \wedge a_1$$

we get the result “true”. So there is at least one configuration such that A can be compiled as module.

4.2 Automatic Listing of Configuration Variants

System builders or also sometimes users have very specific requirements for their Linux Kernel. E.g. one wants to know which ISDN cards work together with which network devices, meaning their

drivers can be compiled (or loaded as module) simultaneously into the kernel.

We can encode our problem as parametric SAT (PSAT) problem [7]. In contrast to a SAT problem, where the input formula is (implicitly) fully existentially quantified, we allow unquantified variables in PSAT. The output is then no longer “satisfiable” or “unsatisfiable” but a propositional formula in the free (unquantified) variables. In [7] we presented a procedure that yields a propositional formula in DNF as output. Each minterm of this DNF describes one possible assignment of the free variables such that the formula is satisfiable under this assignment. So each minterm describes one possible configuration variant of the unquantified options such that the configuration is valid.

We can use this approach and existentially quantify the propositional variables of all options we want to hide from the result. E.g if we want to know all ISDN cards playing together with a current configuration, we quantify all variables except the ones representing the configuration options for the ISDN cards. The output describes then all possible combinations of ISDN cards, that work together with the current configuration.

Example 7 *In the last example we have seen, that A cannot be compiled into the kernel but can be compiled as module. Now we want to know all possible option assignments of B such that A can be compiled as module. We do not care for assignments of the symbols X and C . We check with PSAT:*

$$\exists x_s \exists c_0 \exists c_1 \exists a_0 \exists a_1 \left(\bigwedge_{t \in \mathcal{C}_O} t \wedge \bigwedge_{d \in \mathcal{C}_D} d \wedge \bigwedge_{s \in \mathcal{C}_S} s \wedge a_1 \right)$$

PSAT yields the minterm $m = (\neg b_0 \wedge b_1)$ as result. To satisfy m we need the assignment $\{b_0 \leftarrow 0, b_1 \leftarrow 1\}$ corresponding to the option assignment $B \leftarrow 1$. So in order to compile option A as module, we need to compile B as module.

We shortly explain this result wrt. the configuration file $f(b)$: If $\alpha_O(B) = 2$, $\beta(!B) = 0$ and because of the line depends on $!B$ in the configuration block of A , 0 is then an upper bound for A and thus it cannot be set to 1 anymore. If $\alpha_O(B) = 0$, the upper bound for A is 2, so there is no problem, but since option C depends on B (surrounding if) the upper bound for C is 0 then. But if we set A to 1, we also set the lower bound of C to 1 (because of the select statement) and we have a conflict.

5 EXPERIMENTAL RESULTS

We have implemented all algorithms presented in Section 3. Table 3 summarizes our results for the Kernel version 2.6.33. We state the hardware architecture and the cardinality of the sets $\mathcal{C}_O$, $\mathcal{C}_D$, $\mathcal{C}_S$ and $\mathcal{C}_C$. The last two columns state the number of variables and clauses of a CNF encoding of the resulting formula.

6 FURTHER RESEARCH

This paper takes the first step towards a complete propositional formalization of the LKC. As stated above we consider here only a simplified version of the KConfig files. Not all of the configuration options are visible to the user; Some are only visible under certain circumstances. We want to upgrade our current prototype implementation to an application allowing users to easily create, alter and complete their kernel configurations. Therefore we have to take the visibilities and default values of symbols into account.

Table 3. Experimental results with kernel version 2.6.33

architecture	$ \mathcal{C}_O $	$ \mathcal{C}_D $	$ \mathcal{C}_S $	$ \mathcal{C}_C $	# vars	# clauses
alpha	5931	36	14525	3132	12807	227610
arm	6844	64	15630	4262	14249	246368
avr32	5971	49	14609	3166	12874	227367
blackfin	6371	68	15018	3156	13597	230667
cris	3981	62	9097	1520	8772	136834
frv	5904	41	14533	3135	12769	227587
h8300	3467	33	8034	1473	7692	112149
ia64	6049	43	14775	3201	12999	230185
m32r	5912	38	14537	3141	12763	227826
m68k	5874	35	14468	3125	12793	225735
m68knommu	5950	39	14553	3135	12821	226718
microblaze	5863	38	14449	3123	12660	225132
mips	6321	53	14762	4026	13463	231300
mn10300	5916	43	14543	3143	12788	226915
parisc	5922	39	14551	3139	12783	227301
powerpc	6403	46	14997	3634	13558	234287
s390	5905	39	14468	3177	12739	225791
score	5830	38	14422	3122	12600	224786
sh	6149	51	14762	3326	13154	231231
sparc	5951	40	14571	3169	12811	226660
um/x86	2769	23	6194	1098	6051	102684
x86	6297	43	14978	3250	13402	250430
xtensa	5880	37	14521	3136	12731	226845

We want to systematically check the configuration files for redundant and necessary options as described in Subsection 4.1. Another interesting question is, how many valid configurations there are for one specific hardware architecture. We recently counted models for valid car configurations of the Daimler-Benz car lines [2]. We want to apply these methods to our new configuration problems.

As a long time goal we want to integrate these methods into the Linux build process. Some developers already expressed their wish for tools helping them writing and organizing their KConfig files.

7 ACKNOWLEDGEMENTS

We want to thank Martin Steghöfer and Andreas Kübler for their valuable comments and corrections.

REFERENCES

- [1] Daniel P. Bovet and Marco Cesati, *Understanding the Linux Kernel*, O'Reilly Media, 3rd edn., nov 2005.
- [2] Andreas Kübler, Christoph Zengler, and Wolfgang Küchlin, ‘Model counting in product configuration (to appear)’, in *Proceedings of the Lo-CoCo 2010*, (2010).
- [3] Hendrik Post and Carsten Sinz, ‘Configuration lifting: Verification meets software configuration’, in *Proceedings of the ASE 2008*, 347–350, IEEE Computer Society, Washington, DC, USA, (2008).
- [4] Julio Sincero, Horst Schirmeier, Wolfgang Schröder-Preikschat, and Olaf Spinczyk, ‘Is the linux kernel a software product line?’, in *Proceedings of the SPLC-OSSPL 2007*, (2007).
- [5] Julio Sincero and Wolfgang Schröder-Preikschat, ‘The linux kernel configurator as a feature modeling tool’, in *Proceedings of the SPLC 2008*, eds., Steffen Thiel and Klaus Pohl, 257–260, (2008).
- [6] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, ‘Formal methods for the validation of automotive product configuration data’, *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 17(1), 75–97, (feb 2003).
- [7] Thomas Sturm and Christoph Zengler, ‘Parametric quantified SAT solving (to appear)’, in *Proceedings of the ISSAC 2010*, ACM, New York, NY, USA, (2010).

Declarative Modeling at SAP Revisited – Lessons learnt when Applying Configuration Techniques

Albert Haag

Abstract. We have had the opportunity to make an explicit comparison between a rule-based and a constraint based implementation of an internal SAP application that serves as our case study: “Scope Definition for an SAP® Business ByDesign™ Solution”, a complex interactive configuration of a business software system. We compare the behavior of the two implementations and identify the pros and cons of a declarative approach and summarize our experiences. In the case study there were strong advantages in moving to declarative processing. Nevertheless, arguments for retaining many procedural elements also prevailed. The body of this paper is an extended abstract in advance of a more complete treatment that is in preparation. It is meant as a basis for discussion at the workshop, where, if possible, we plan to give a live demo of the new (declarative) implementation.

1 Introduction

The SAP® Business ByDesign™ Solution” contains an application that allows a user to interactively configure (customize) the system from the business perspective. This application is called SAP® Business ByDesign™ Scope Selection. It is based on a catalog of business packages, topics, and options that represents the solution capabilities of a SAP Business ByDesign solution. The user can select (or deselect) items from this catalog. The resulting set of selected items must satisfy given business constraints. A valid selection represents the intended solution scope of the user and is the basis for a subsequent non-interactive completion of low-level customizing of the system.

In the previous version of SAP® Business ByDesign™ Solution” Scope Selection was implemented as a rule based system (which we consider to be procedural). For the upcoming version we had the opportunity to replace this rule-based engine with the core of our declarative configuration engine. We wish to stress that the result is a real productive application, not a prototype.

The project stipulated, however, that the rules were not to be re-modeled. This means that the run-time behavior of the system changed, not the underlying product model. This was a good decision that contributed to the success of the project as it was the enabler to meet some practical constraints that are often overlooked in theoretical analyses. An existing knowledge base of business rules (or constraints) represents an enormous investment. A change of paradigm means modifying (and validating) the application for maintaining the rules/ constraints, retraining personnel, not to mention re-aligning the various groups involved on any change to any rule/ constraint. This takes time and will be undertaken only if the criticism of the existing approach is so severe that a paradigm shift is mandated. This was not the case here. The shift in paradigm was more a by-product of re-using our existing configuration engine where otherwise a re-implementation of the rules engine would have been needed to conform to new internal technical constraints.

Also the rules were in a general form

IF «condition» **THEN** «value1» [**ELSE** «value2»]

Where «value1» and «value2» refer to the selection or deselection of a business item and «condition» is a logical expression whose atomic terms also refer to the selection or deselection of business items. This general format allowed a direct interpretation as logical inferences. It should be noted that it is also possible to specify static and dynamic defaults. By a static default we mean designating an item in the catalog as selected by default. By dynamic defaults we mean a rule that selects an item as a default when its «condition» applies.

Another interesting observation from our case study is that the people responsible for a specifying a configuration application (the “solution managers”/ “product owners”) often tend to favor a more procedural approach and need to be convinced of the potential benefits of switching to a declarative approach. There are again reasons for this. The solution managers are responsible for the look and feel of the application and its ultimate acceptance by users. They are not convinced that the normal user can understand or deal with situations that arise from purely declarative processing. An example: The business rule

IF «A.selected» **THEN** «B.selected»

(where A and B refer to some items in the catalog) is logically equivalent to

IF «B.deselected» **THEN** «A.deselected»

Nevertheless, the former (an explicitly maintained business rule) may be deemed to make perfect sense to a business person, whereas the latter may be deemed to be unintelligible to the same. (This is more an example against being logically too complete than for a direct procedural flow of control. Arguments for procedural flow of control ensued from explicit specifications of the system behavior that do not follow declaratively from the knowledge base, i.e. attempting to help the user with some non-declarative heuristics – automatic auto-completion in word processors is of this type.)

2 Declarative vs. Procedural Processing

It is generally accepted that a declarative approach to modeling configurable products is superior to a procedural one. The main benefits of declarative processing at run-time are in the area of explanation and user guidance as well as coping with an unforeseen flow of control.

An unforeseen flow of control typically happens when the user makes arbitrary changes to an existing configuration, but the intended order of user decisions suggested and proscribed by the UI was designed for making an initial configuration. For example, it may be assumed that the user will initially decide on their selections by making decisions in a proscribed order (e.g. processing the catalog in a strictly top-down fashion). The procedural control flow is usually designed with this order in mind. But very often there are unwanted side-effects if the user chooses to make decisions in a different order. Thus the UI may actually enforce the proscribed order of decisions (this is not quite the case in our case study). However, when designing the procedural control flow it is difficult

to foresee all situations that occur when a user modifies an existing configuration.

There are also some problems also need to be overcome in a declarative approach. We shall look at one particular problem: the representation of a user choice in the configuration UI as a checkbox which allows only two values (selected or deselected). The UI state *deselected* cannot distinguish *intentionally deselected* by the user from an initial state *unspecified* (unknown) – where neither the user nor a constraint have yet reached any conclusion about the item. Again general UI guidelines that need to be observed may dictate that there be only two states. The solution managers may again feel that a normal user would not understand three states or consider a UI with three states unnecessarily complicated.

This poses a challenge for a purely declarative processing. There are some constraints in the form $A \vee B \vee C$ that stipulate that at least one of the items *A*, *B*, or *C* must be selected. A declarative engine should detect and report such an *incompleteness*. A simple approach would be to initially treat all items as de-selected by default. However, this is counter-productive. It leads to many irrelevant inferences, a multitude of irrelevant conflicts between defaults, and generally clutters up explanations. It also poses an unnecessary performance challenge for our architecture. Hence, a mechanism is needed to uncover *incompletenesses* while otherwise avoiding inferences from unfounded de-selections in an intermediate state of the configuration. (Of course, when the configuration is finished all items not explicitly selected are treated as de-selected.)

The new declarative implementation of the Scope Selection application provides advantages both for the modeling environment and for the runtime behavior.

2.1 Advantages for Modeling

As stated the rule base was not changed. The existing product model was reused without the need to change rules. This was actually a precondition for the successful transition from the previous implementation to the new implementation and increased the acceptance of the new Scope Selection application by the solution managers. However, the rules are now interpreted as logical inferences (constraints). This has several hidden benefits (for future versions of the application):

1. It simplifies the future maintenance of rules (constraints) because the developer does not have to worry about procedural interaction with rules defined by other developers (or the order in which rules are applied).
2. It is now possible to formulate rules that obey the original syntax but will be interpreted in a more logically complete way. This would make it easier for partners to extend the rules when providing their own business add-ons
3. Rules can be simplified because it turns out that their condition often contains logically redundant elements needed only for guaranteeing correct procedural processing
4. There is a possibility to implement generic constraints such as that the selection of an item implies the selection of the parent.

2.2 Advantages at Run-time

In the current version the ability to retract user choices consistently was the basis for multi-step undo/redo functionality in scope selection. Furthermore a simulation function has been implemented which can be used to check before a scope change the impact on other scoping elements. Also the ability to allow user input in unforeseen order is an important benefit of the paradigm change to a declarative approach.

The handling of scoping conflicts and resolving inconsistencies is a new topic in the Scope Selection application. The previous implementation rather overruled user decisions in case of conflicts which could sometimes lead to unpredictable changes of affected scoping elements. Now, by using a declarative constraint engine, conflicts are first reported and then resolved by the user. This principle ensures that the resolution of inconsistencies really reflects the intention of the user. Even where procedural control flow is added on top of the declarative processing the user gets a clear feedback from the system and about which of the decisions have to be retracted to solve a conflict, and in combination with the multi-step undo/redo the user can easily go back to the previous scope configuration state.

3 Specific Features of Our Declarative Approach

We briefly describe our configuration architecture and two extensions to it that came out of this project. (We delegate further details to our more extensive treatment that is in preparation.)

For the case study the (A)TMS functionality of our engine was the key feature. This combines a Justification Based Truth Maintenance System (see [2]) with add on calculation of ATMS labels (see [1]), and some own extensions (see [3]). The aspect of generating explanations and the role of an ATMS therein is dealt with in [4].

The (A)TMS was extended to deal with the following:

1. Declarative default handling – when and how can the configurator automatically override a default in a declarative way. By this we mean that the outcome is the same for any order in which the defaults are made. In particular, if the configuration is saved and later reloaded (without the entire (A)TMS state) the configuration must be identical before the save and after the re-load.
2. Declarative way of handling *unspecified* (incompleteness detection) – when and how can the system detect an incompleteness in a declarative way – without resorting to means such as treating unspecified items as deselected by default.

4. Procedural Elements Retained for the Time Being

As we have already stated the new Scope Selection application retains some procedural logic despite the advantages of a purely declarative approach. This has been done as a kind of practical compromise to preserve the look and feel of the existing Scope Selection user interface as much as possible and to avoid a major change in the user interaction for the Scope Selection process. The emulation was performed outside the core engine, which was not modified to fulfill these requirements in any way. However, this weakened some of the advantages of the declarative approach listed in section two above.

The procedural elements fall into two main categories, which we briefly outline in the following subsections.

4.1 Guidance of the User along the Catalog Hierarchy

The product owner requested support for guiding the user to make scope decisions top-down along the hierarchy of the catalog. Hence the user has to decide first about a business package before they can select any of the sub-topics. For example, the business package “*Selling Products and Services*” represents the overall functionality to support the entire process of customer orders, from sales order entry to order confirmation and delivery. The key features are

modeled as sub-topics, e.g. “*Sales Orders*” and “*E-Selling*”. So, first the user has to decide whether to select the business package “*Selling Products and Services*” or not. The user is only able to select “*E-Selling*” once it has been selected.

In principle, this initially helps the customer to focus on the main functionality and reduces the number of possible items that have to be looked at. However, there is the drawback that the user is restricted in their freedom of choice (they may know that they primarily need “*E-Selling*”). Also, many special situations need to be considered carefully, such as when a user first selects the package, next sub-topics are selected by the user or by constraints, and then the user would like to take back the selection of the package. It may be hard to manage these completely in practice.

4.2 Automatic Resolution of Conflicts

The engine provides support for declarative resolution of conflicts (declarative default handling). Nevertheless, the solution managers/product owners requested that the system resolve conflicts due to some heuristics, even when this is not justified declaratively. These heuristics are implemented by procedural means. In some cases, even user decisions are retracted by the system. However, the user is informed if this happens and can use the undo feature to revert to a previous state.

We give one example of such an automatic conflict resolution. There are different business options to choose which payment methods should be used for outgoing payments. The possible options are “*Bank Transfers*”, “*Wire Transfers*”, and “*Checks*”. The following rule addresses these options:

IF not “*Checks*” **AND** not “*Wire Transfers*”
THEN “*Bank Transfers*”

The intention of the developer who defined this rule was to ensure that the payment method “*Bank Transfers*” is automatically selected in the absence of the other two. Now, provided that the user de-selects all three options this would result in an inconsistency, which cannot be resolved by declarative means (it involves three choices all directly attributable to the user themselves). However, by interpreting the above rule as a heuristic to use in resolving such a conflict the user decision to deselect “*Bank Transfers*” is overridden; this option is selected on the user’s behalf and the user is not faced with the task of dealing with the inconsistency.

However, the problem with such mechanisms is that they work fine in many of the foreseen use cases, but may cause confusing behavior in other unforeseen use cases. Therefore we don’t recommend this approach. In the specific example it would be better to define “*Bank Transfers*” as a default. This would cover most use cases involving the potential incompleteness. If the user intentionally deselects this option and thereby causes a conflict or incompleteness, they need to handle it themselves.

5. Summary

In summary, we conclude that for the case study the declarative approach is beneficial over the procedural approach. However, there are reasons why one often cannot stick to a purely declarative approach. Some are due to practical constraints as outlined above for our case study. In particular, we mention again that it is not always considered a boon to strive for logical completeness (or at least the “real” user intelligible explanations need to be separated from ones that may be logically true, but are meaningless to the user.)

1. Some problems are not declarative by nature (e.g. slot allocation)

2. The particular constraint framework used may be lacking in expressiveness (i.e. it is not possible to easily express certain things declaratively, although this could be done given a more powerful constraint language. (Recall that configuration problems are often not classical CSPs so additional features are required. For example, in engineering applications floating point variables with interval domains are common.)
3. If the overall configuration task is structured as a hierarchy of sub-problems, one often desires only a downward flow of information in the hierarchy. Upward or sideways flow is inhibited (also for performance reasons).

REFERENCES

- [1] Johan de Kleer, ‘An assumption-based truth maintenance system’, *Artificial Intelligence*, 28, 127–162, (1986).
- [2] Jon Doyle, ‘A truth maintenance system’, *Artificial Intelligence*, 12, 231–272, (1979)
- [3] Albert Haag, ‘Sales configuration in business processes’, *IEEE Intelligent Systems*, 13(4), 78–85, (1998).
- [4] Albert Haag, Ulrich Junker and Barry O’Sullivan ‘Explanation in Product Configuration’ in *IEEE Intelligent Systems*, January/February 2007

Software Product Lines

Lessons learned when applying configuration techniques

Marcos Didonet Del Fabro and Patrick Albert¹

Abstract. Software Product Lines (SPL) aim at creating a model of software that shares a set of interrelated features, which will be combined in different ways to create sets of products. The problem of composing the right set of features can be seen as a configuration problem. Most part of existing solutions of SPL are based on generating relatively simple boolean predicates for product validation. In this short paper, we explore two design alternatives that apply configuration techniques to solve SPL problems. The objective is to identify the advantages and drawbacks of each approach, and to verify if the same framework (modeling artifacts, constraints and solver) can be efficiently used for implementing the solutions. First, we define the shared software model using concepts of the domain, in a configuration-like manner. Second, we defined the shared model using a more abstract representation, widely used in SPL, called *feature models*, which is an intermediate representation before the final product representation. We present the first lessons learned, as well as possibilities for improvement.

1 INTRODUCTION

A Software Product Line (SPL) [7] is a software system that shares a common set of features of an application domain, which acts as a basis to produce a set of products. The specificities of each product are defined by features satisfying the needs of a particular application. These features contain explicit variation points and constraints, which will guide the production of the final products. Each combination of variation points may be used for producing a distinct product. For instance, consider we need to generate software that runs on mobile phones. One feature could represent the presence (or absence) of digital camera. In this case, the product line would generate two solutions, one with digital camera handling capabilities, another one without. This problem can be generalized as a software configuration problem, i.e., that is given a generic specification of a software model, plus a set of constraints over this model, produce one, all or the best software models that satisfy these constraints.

There are different solutions coupling product lines and configuration. For instance, [8] provides a conceptualization of configuration models synthesizing existing configuration approaches. In [1], on the other hand, the solution uses feature models for producing decision trees. The difference between both formalisms and frameworks makes the comparison difficult. From the best of our knowledge, this is the case for the majority of the solutions. In this short paper, we explore and compare two different alternatives for representing and solving SPLs. In the first approach, the goal is to produce a model of the final product(s) that respect the input constraints. This

is similar to existing configuration solutions, where the main goal is to find/produce a representation of a product for an application domain. The main difference is that the product that is configured is a software, i.e., classes, methods or fields. In the second approach, the goal is to produce all the combinations of *feature models* [2], which is the most used approach on existing SPL solutions. The objective is to identify the advantages and drawbacks of each approach for designing SPLs and to verify if the same framework (modeling artifacts, constraints and solver) can be efficiently used for implementing the solutions.

We present two simple examples using each approach, and we finalize by comparing both approaches, with the main advantages or drawbacks for using one or another alternative. The central conclusion from our experiments is that configuration techniques can be applied to Software Product Lines, and more research and implementations should be conducted towards this direction.

2 CONFIGURATION APPLIED TO SOFTWARE MODELS

In this approach, the goal is to produce a model that is close to the final product(s), i.e., a **software** or a set of softwares satisfying the domain constraints. We have implemented a generic framework using Model Driven Engineering techniques for software configuration, i.e., all the artifacts are described in terms of terminal models, meta-models and operations between them [3]. The metamodels describe the concepts of the domain; the terminal models describe the studied objects (the instances). There are different kinds of operations, such as transformations or merging. The constraints are written in a variation of the OCL (Object Constraint Language of OMG) [6]. The configuration operation, i.e., the production of a terminal model respecting a set of constraints has been encapsulated into an operation called *ModelSearch*. We implement all the components as plug-ins for the Eclipse platform. The problem specification is translated into the OPL Language [9], which is executed in the IBM ILOGTMCP Optimizer engine. The whole approach explained in details at [4].

Consider for instance a SPL for producing software executed in watches. We want to generate 5 different kinds of watches: 1) one simple watch, 2) one with alarm, 3) one with sound alarm, 4) one with sound and visual alarm and 5) one with visual alarm. In other words, the metamodel define classes, attributes, methods, etc. to handle watches. There are different ways of modeling such problem. For instance, we could have one class per kind of watch, or have a unique Watch class with different sets of methods. We have chosen the second one, because it is simple and concise. We provide below a simple metamodel (we use a Java-like notation):

```
package watches {
```

¹ IBM France, ILOG Research Projects, email: marcos.ddf, albertpa@fr.ibm.com

```

class Root {
  reference classifiers[1-10] container : Class;
}
abstract class Class {
  reference methods[1-15] container : Method;
}
class Watch extends Class {
}
abstract class Method { }
class DisplayTime, Start, StartAlarm,
  StartSoundAlarm, StartVisualAlarm,
  Stop, StopAlarm extends Method { }
}

```

However, not all combinations of methods are allowed. The derived models should respect the following constraints, which are implemented using an OCL-like language.

- the *DisplayTime* and *Start* methods are mandatory;
- if there is a *Start*, there is a *Stop*;
- if there is a *StartAlarm*, there is a *StopAlarm*;
- if there is a *StartSoundAlarm*, there is a *StartAlarm*;
- if there is a *StartVisualAlarm*, there is a *StartAlarm*.

3 CONFIGURATION APPLIED TO FEATURE MODELS

In this approach, the goal is not to produce the final models, but to produce valid combination of features, i.e., *feature models*. We chose this format because it is the most common format in SPL solutions (it has been first introduced at [2]). The metamodel of feature models defines a tree-structure, with a root feature which contains 1 to N children features, which may contain themselves children features. The features may be interrelated, for instance, one feature may require another feature, or two features may be mutually exclusive. The features may be grouped with minimum and maximum cardinalities. Such specification is generic enough to be used in distinct domains. Each application will have a set of different interrelated features. We show a simplified version of the metamodel below. One of the key characteristics of this representation is the independence from the final software that is generated. For instance, one single feature may generate one or several output elements. This requires to create a mapping from the feature model into the product representation (e.g., to map one feature to a class). However, this mapping is out of the scope of this paper.

```

package FM {
class FeatureModel {
  reference root[0-1] container : RootFeature;
}

class Feature {
  attribute isMandatory : Boolean;
  attribute name : String;
  attribute isSelected : Boolean;
  reference children[*] container : Feature;
}
class RootFeature {
  reference children[*] container : Feature;
}
class Group extends Feature {
  attribute minCard : Integer;
  attribute maxCard : Integer;
}
}

```

Consequently, the domain model differs on its nature: it is not the model of the product that is configured, i.e., the final output is one or more feature models, each of them containing a different set of valid features. This specification can be used for modeling any kind of product line. Additional constraints for each different SPL should be added using a constraint language.

We have used the same framework that was used to implement the example from the previous section. In addition, we have imported a set of examples from the SPLOT feature model repository (see [5]). The feature model shown in figure 1 conforms to the metamodel presented above; it is a product line for configuring a car (*Car PL example*). The implementation and execution of such example has produced 9 valid output feature models.

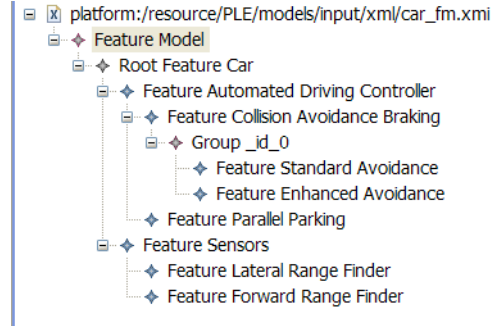


Figure 1. SPL model extracted from the SPLOT repository

In addition to the feature model, we also have defined two inter-feature dependencies. As in the previous use case, they are implemented using an OCL-like language.

- if the *EnhancedAvoidance* feature is present, the *ForwardRangeFinder* feature is also present;
- if the *ParallelParking* feature is present, the *LateralRangeFinder* feature is also present;

4 CONCLUSIONS AND LESSONS LEARNED

We implemented a generic configuration framework based on Model Driven Engineering techniques, and we applied it to software product lines applications. We explored two different modeling alternatives, using the same framework and both solutions have proven to be feasible.

In the first approach, the goal was to produce model(s) of the software(s) that are being configured, i.e., a set of classes, methods or attributes. We have considered that all the elements are variable. This approach has shown some advantages. First, we may reuse the ideas of the large number of works on configuration, that have proven to be efficient in different application domains. We reason over the domain objects, and the produced model can be used on other applications. In addition, it is relatively simple to produce a mapping between the modeled problem and the final solution.

In the second approach, we have modeled the problem using feature models, which is the most common approach for Software Product Lines. We were able to define the problem using the same framework, without the need to adapt it. However, instead of producing a model of the product, the result is a set of interrelated features. The independence from the model of the product (the software that

will be produced) enables to easily specify features without taking into account implementation details. One drawback is the necessity of having an extra step of establishing the correspondences (mappings) between the feature models and the actual software model that should be generated. We have also seen that feature models have simple constraints: it uses logical connectors (AND, OR), cardinalities, plus implications between the features. The only attributes/elements that need to be configured is the *selection* of features. There is a risk of *overmodeling*, i.e., that is using a very rich CP-based framework while we only need a couple of simple constraints. On the other hand, it opens the possibility of studying the implications of having more complex constraints, i.e., if they are useful, how to write them, etc.

To summarize, we were convinced based on our experiments that Software Product Lines can be seen as software configuration problem, where the application domain is the software itself. We notably stress the usefulness of starting with the examples from the SPLOT repository, which are a good initial point for experimentations. We plan to use our framework in other models from this repository, and to apply both approaches to the same specifications. This will improve the comparison of the alternatives presented. However, it is necessary to implement a mapping from this repository into an object oriented format, which is itself a subject of study. We suggest that other researchers use this repository as well, to ease the comparison between approaches.

We were able to see the major differences between both approaches, but we expect that more experiments will enable to increase our understanding of the problem and to improve the framework as well.

ACKNOWLEDGEMENTS

This paper has been partially funded by the IdM++ ANR French project.

REFERENCES

- [1] L. Hotz, K. Wolter, T. Krebs, J. Nijhuis, M. Sinnema, S. Deelstra, and J. MacGregor, *Configuration in Industrial Product Families - The ConIPF Methodology*, AKA-Verlag, 2006.
- [2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, 'Feature-oriented domain analysis (foda) feasibility study', Technical report, Carnegie-Mellon University Software Engineering Institute, (November 1990).
- [3] Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez, 'Model-based dsl frameworks', in *OOPSLA Companion*, eds., Peri L. Tarr and William R. Cook, pp. 602–616. ACM, (2006).
- [4] Marcos Didonet Del Fabro Mathias Kleiner and Patrick Albert, 'Model search: Formalizing and automating constraint solving in mde platforms', in *ECMFA 2010*. Springer, (2010, to appear).
- [5] Marcilio Mendonca, Moises Branco, and Donald Cowan, 'S.p.i.o.t.: software product lines online tools', in *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pp. 761–762, New York, NY, USA, (2009). ACM.
- [6] *OCL 2.0 specification*: <http://www.omg.org/spec/OCL/2.0/>, 2008.
- [7] Clements P. and Northrop L., *Software Product Lines: Practices and Patterns*, Addison-Wesley Professional, 1st edn., 2001.
- [8] Juha Tiihonen, Timo Lehtonen, Timo Soininen, Antti Pulkkinen, Reijo Sulonen, and Asko Riitahuhta, 'Modeling configurable product families', in *Delft University of Technology*, pp. 1139–1142, (1998).
- [9] Hentenryck P. V., *The Optimization Programming Language*, MIT Press, 1999.

Structured Development Process of Configuration Models

Plietz, Matthias¹

Abstract. Current market situation demands highly variant customer individual products in engineering, automotive, component manufacturing and other branches. Adaptivity along customer requirements will become a unique selling point for European companies. The control of product variance as well as the administration and maintenance thereof become substantial aspects of variant management.

This paper should propose an approach for a more effective management of the product knowledge.

1 INTRODUCTION

The management of configuration knowledge is used to handle variants by effective management of building sets (internal variance) and by definition of options to be presented to the customer (outer variance, see also [9]). By the help of configuration systems the greatest possible outer variance should be represented in conjunction with minimal internal variance.

A global approach for data maintenance and management is necessary to handle the knowledge and logic of the complex products. This global view integrates

- all divisions of a company regarding to value chain,
- all kinds of knowledge (unspecific, asking, describing knowledge) and
- the life cycle of the product.

The data maintenance of complex configurable products is not only a process of modelling. It should be used by all involved divisions of the company to retain information. It must be comprehensible (not only for computer scientists) and must support collaboration between involved divisions.

2 KNOWLEDGE MANAGEMENT PROCESS

2.1 Knowledge abstraction levels

In practical work the maintenance and management of the product knowledge is split into several abstraction levels

- the expert's knowledge about internal relationships, chains, contextual requirements and needs and
- the coded model knowledge in the syntax of the used configurator

Scheer [17], Sinz [19] and Ghoffani [8] follow this approach of splitting.

The expert knowledge will be generated and changed from implicit into explicit information by the techniques of knowledge acquisition. Schienmann describes in [18] important methods to acquire knowledge. Bretoni and Schneider point out the change process of explicit knowledge (in [3]).

The knowledge of an expert is rather unstructured and wide independent of the syntax of the used configurator system. It contains mainly

- the knowledge of specialists (such as engineers, designers, product managers)
- options and limits of the product (defined by product manager)
- applied information and experiences
- ideas, requests, requirements (i.e. by sales)
- product structure

The coded model knowledge will be retained by the configurator editor in the concrete code syntax of the used configuration system. It combines the content of knowledge of the experts with the syntactic structures of the configurator.

2.2 Sequence of processes

The conversion process can be described as the transition of expert into model knowledge. This process contains the concrete phrasing and formal storage of knowledge in the syntax of the configurator.

Ghoffani [8] provides an extensive aggregation list of important methods to acquire information. He points out the necessary support by a knowledge manager. This declaration is similar to other fields of knowledge management (see [15]).

Hence the expert in most cases needs a "translator". Figure 1 describes the process of knowledge acquisition and transition. Thus it appears that the expert as well as the knowledge manager should work cooperatively and synchronizedly to achieve the general result.

This process has some consequences and disadvantages:

1. The knowledge manager acquires the expert knowledge without enforcing of completeness and consistency.
2. The knowledge manager translates the expert knowledge into the syntax of the configurator synchronously.

¹ ORISA Software GmbH Jena, Germany, email: Plietz@orisa.de

3. The expert is not able to understand the syntax of configurator, so he can only check the acquired and implemented knowledge by checking the running configuration model itself.
4. This validation can only be done phenomenological in particular cases; a full-blown test is not feasible.

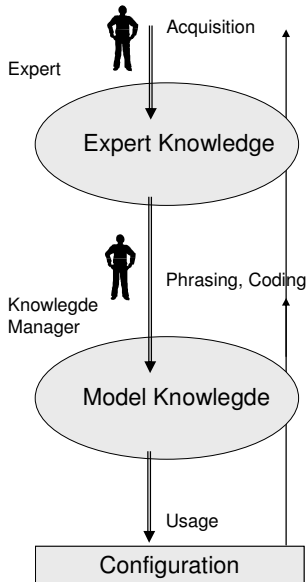


Figure 1. Phases of knowledge processing

2.3 Cycle of processes

Mertin et.al. [14] interpret the process of knowledge management as a cycle of single phases. Such an approach is described as Fraunhofer Knowledge Management Audit (FKMA) in [14]. Also the processes of acquisition and processing of product knowledge can be characterized as a circular process containing generation, transition and usage of information.

Figure 2 uses the FKMA phases to describe product knowledge management.

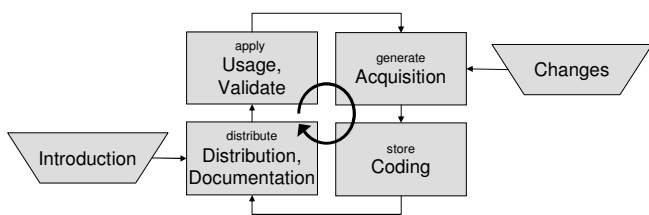


Figure 2. FKMA process adapted to product knowledge management (see [14])

2.4 Superposed processes

To avoid the disadvantages of the sequential approach we propose to instantiate two separate processes of knowledge management. These processes should be superposed and collaborate by references and mappings. Applied to the management of product knowledge of configurators the expert knowledge and the model knowledge must be separated consequently.

In the field of software engineering, this approach is described as the paradigm of model driven architecture (MDA, see [10]). Sinz [19] already used this paradigm to solve a limited and well defined configuration task.

The generalized structure of management of the product knowledge enables the handling of two independent circular processes. These processes are assigned to the abstraction levels expert and model knowledge and interact with assigned data in separate phases. The information flow between the abstraction levels is restricted and defined.

Figure 3 points out the combination of sequential and circular process approaches to maintain and process the product information by the help of FKMA (see [14]).

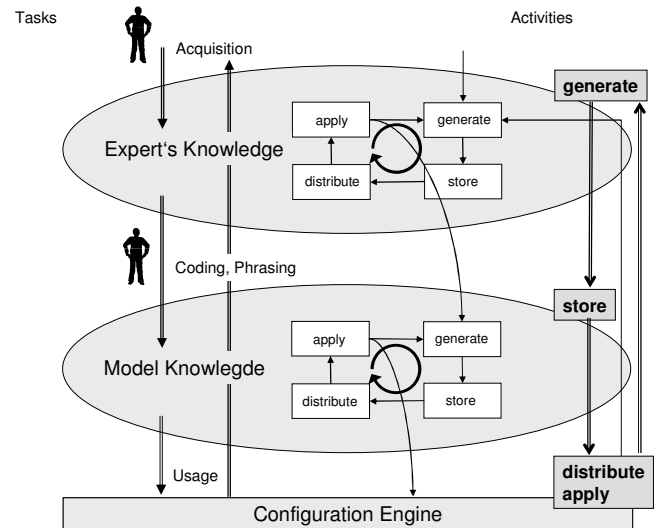


Figure 3. Combination of sequential and cyclic process approaches

This approach forces the splitting of the general process into demand and realization and follows the common sense of today's software engineering.

It allows adequate context-related and separate validation of the expert knowledge and the coded model knowledge. This approach conforms to work flows in a company and offers suitable levels of understanding for several tasks as well as several divisions of the company. It enables cooperative and asynchronous work of several divisions.

The split into two processes enables maintenance, validation, completion and documentation of expert knowledge by the experts themselves. The distinct data storage and processing phases are supported by the split processes. Hence the knowledge manager must support the experts only in complicated situations to acquire information. Asynchronously he has to transform and phrase the expert's information into the model knowledge to code it according to the configurator's syntax.

3 FORMALIZING OF KNOWLEDGE

The process to maintain the model knowledge uses the code base, syntax and semantics of the concrete configurator engine. The ramifications for the storage and maintenance of the expert knowledge and its transition will be described in the following.

3.1 Type and range of expert knowledge

The following requirements have to be considered before choosing structures for display and maintenance:

1. The system should allow the phrasing and storing of partial and diffuse information.
2. Also non-computer scientists should be able to maintain and administrate the knowledge in an understandable manner.

To solve these generalized demands a formalized language must not be used. Experience shows: Either such a language is easy to learn but inexpressive. Or a language is more expressive but requires a huge effort to learn and to use. Both cases are not suitable as a solution for non IT experts. To solve these demands the expression of the structured prose (see [1]) will be proposed. This expression uses natural language to phrase facts, relations and ideas. There are minimal requirements for an IT system concerning structuring of the knowledge. In such a system the knowledge should be separated and handled by entities to support maintenance and administration. It requires minimal structuring by names, classifications, parameters and links to subjects and documents.

Schienmann [18] and Fowler [5] supported this approach, but from different points of view. Also Ghoﬀani supposes that the natural language is a common and available tool to express information. Issues, facts and circumstances can be expressed and phrased in high density ([8]).

In contrast, Sinz proposed to provide a specific language (in today's meaning a "domain specific language" [DSL]) in a specialized context and domain. In [19] he described a method to phrase the demands of the experts and to convert it into the configurator code automatically. Because of the narrowly defined domain this approach can not fulfil the requirements of a generalised knowledge management.

3.2 Partitioning of expert knowledge

3.2.1 Structuring by knowledge entities

Based on the demands of the knowledge management the formal processing requires splitting and separation of the expert knowledge into meaningful knowledge entities (see [15]).

So the general topic of the expert knowledge should be split into useful amount of single subjects. Each subject will be connected with a knowledge entity. In practical work this process of structuring is a periodic process to add, change and update information and their structure.

The type of such an entity should be defined non-specific and independent of the content. So these entities will be able to store single subjects by the paradigm of structured prose and will be parameterized by defined criteria. To use such entities in the domain of product modelling we defined base types of entities. These base types are defined context specific [4] and adapted to the demands to phrase the expert knowledge.

The paradigms of the model-view-controller (MVC) concept (see [6]) were adapted to construct these base entity types. The external, user interacting entities as well as the internal, data processing entities are defined according to MVC. The controller task will be assumed by relationships between entities.

Our defined base types are displayed in Table 1.

Table 1. Knowledge subjects and entity types

Knowledge and information about:	Base type of Knowledge entity	includes:
Behaviour and relations	Fact	Relations, constraints, tables, formulas, documents, demands, issues, ideas
Appearance, input, output	Interface	User interface, data to other systems
Grouping of subjects	Subject	Thematic capsule i.e. the product, domains, processes
Grouping of processes		
Structure of product		

3.2.2 Cross-linking of knowledge entities

The product knowledge belongs to one general topic independent of the splitting and separation into several entities. Hence the general information about the topic is stored not only as the total of the entities but also as relations between the entities connected with each other.

The approach of net-like associations of related knowledge entities corresponds to the current paradigms of successive knowledge acquisition and management used for other purposes (see also Petersen in [16], Mertins et al. in [15] or Harasymowicz-Birnbach in [11]).

During our analysis we defined simple meaningful types of relation and applied them in the general topic of the expert knowledge. So it would be possible to create and define associations between the knowledge entities to express combined, conditioned, net-like or hierarchical sets of entities.

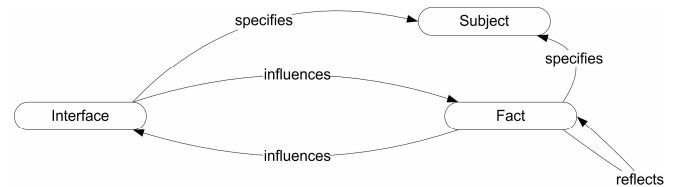


Figure 4. Used knowledge entity types and their relation types

Figure 4 shows the knowledge entity types and their valid relation types by the help of an ontology.

3.3 Structuring by design pattern

3.3.1 Motivation

The described knowledge entities conform to the requirements of structural handling of related knowledge. But for practical work they have some disadvantages:

1. The base knowledge entities may be too simple because of their elementariness and their similarity to the objects of the model code.
2. The base knowledge entities don't contain any predefined structure of partial solutions and aspects and aren't able to reduce the complexity of the expert knowledge.

Unlike entities, design patterns (see [1]) have the potential of simplifying and structuring the knowledge management. Alexander defined the term „Design Pattern“ as a relation between problem and solution:

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution." (Alexander in [2 p.247]).

This approach enables the pre-conditions to define a unified expression for problem descriptions and solutions, for collections of solutions of typical, periodical appearing problems.

Gamma et al. [20] used the pattern paradigm to describe experiences and best practice of object oriented programming. Lee, Eastman & Sacks [13] defined patterns to describe and normalize product models to interface enterprise applications. Kohls [12] uses patterns to structure experience and knowledge of experts and to harmonize procedural methods for collaboration of specialists.

3.3.2 Usage of patterns for configuration models

If patterns are used in the domain of configuration models the advantages of the patterns like simplified maintenance and illustration of the expert knowledge will appear. Reasons are the content driven structure and the reusability of pattern.

So this pattern based approach supports and enhances the approach of base knowledge entities described below. The patterns are more complex than the knowledge entities and contain the content based on their own pattern methodology. From a generalized point of view the knowledge entities become fundamental patterns.

To use the pattern approach in the domain of configuration we define these basic demands:

1. All related persons (specialists, experts) should get the same understanding about the patterns by the help of naming, description, criteria and visualization.
2. The size of model description should be reduced by shifting complexity into the pattern.

3.3.3 Sample pattern

Fundamental patterns are derived from the knowledge entities and can be used for fundamental expressions.

Table 2. Fundamental Patterns

Nr	Pattern	Description
I.	Attribute Pattern	Describes parameter and classification (see [13])
II.	Fact Pattern	Describes logical contexts and influences
III.	Interface Pattern	Describes communication with user (inputs and outputs) and systems
IV.	Subject Pattern	Describes issues and subjects of a topic

Structured patterns are used to group and enhance the information and follow Alexander's paradigm of „enfolding“ (see [1]).

Table 3. Structured Patterns

Nr	Pattern	Description
V.	Specialization Pattern	Problem: Assignment and Encapsulation of subjects Solution: Defines assignments of specializations to subjects
VI.	Follower Pattern	Problem: Definition of coequal and conditional structures Solution: Defines assignments between structures and their relations Usage: Conditioned processing of model ranges, conditioned input/output, conditioned creation of components
VII.	Hierarchy Pattern	Problem: Definition of Hierarchies Solution: Defines assignments of subordinate structures and their conditions Usage: multilevel configuration with main and sub parts, conditioned input/output

Specialized and derived patterns describe concrete situations of the domain, but can be ascribed to more generalized pattern. Samples are described in Table 4.

Table 4. Derived Patterns

Nr	Pattern	Description
VIII.	Claim Pattern	Split sales and manufacturing related demands and information
IX.	Screen Pattern	Describes user interaction and behaviour, generalized input and output

3.4 Transformation of expert knowledge

Because the natural language is used to phrase and store the expert information this knowledge must be transformed into the model code by the knowledge manager manually. This transformation process is done asynchronously to the generation process of the expert knowledge.

It should be processed based on an almost finalized knowledge content. So the knowledge manager will be able to detect incorrect, incompatible as well as incomplete statements. He could then discuss these inconsistencies with the experts.

From the beginning of the transformation process the knowledge manager could determine the most complex knowledge ranges. Complex ranges could be compact statements or clusters of logical associations. According to the greedy approach (see [7]) these most complex ranges can be found and processed. This paradigm forces the solution of the most complex problems at first and enables the subsequent solving of all less complex problems.

During the transformation process the knowledge entities and patterns of the experts will be connected to the code objects of the model knowledge by additional implementation relations. The definition and forwarding of processing state information will support the complete transformation.

4 SUMMARY

This paper should motivate to split the knowledge management processes of configuration information into separate units of expert and model knowledge. We described approaches to structure the expert knowledge. Realized knowledge entities and design patterns will be described in further papers and sessions.

This approach generates essential information by the relations of knowledge entities and patterns. Assignments between expert and model knowledge make enhance

- correctness and consistency

Transformation faults, not transformed entities, inconsistencies and overlappings could be detected.

- understanding and transparency

The analysis of implemented structures concerning dedicated expert knowledge should help to understand the model behaviour. The conclusion back to the reason of implemented structures should support the training and as well as the error analysis.

5 REFERENCES

- [1] Alexander, Christopher ; Ishikawa, S. ; Silverstein, M. u.a.: *A pattern language*. New York, NY: Oxford Univ. Press, 1977.
- [2] Alexander, Christopher: *The timeless way of building*. New York, NY: 24. print. Aufl. Oxford Univ. Press, 1979.
- [3] Bettoni, Marco C. ; Schneider, Sibylle: *Experience Management: Lessons Learned from Knowledge Engineering*. Fachhochschule beider Basel, 2002
- [4] Efftinge, Sven: *Do you speak DSL?* (In German: *Sprechen Sie DSL?*) In: Javamagazin (2009) Nr. 12, S. 14-21
- [5] Fowler, Martin: *Language Workbench*. URL <http://martinfowler.com/articles/languageWorkbench.html>. - 14.3.2010
- [6] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns - Elements of reusable object-oriented Software*. Reading, Mass: 2. Aufl. Addison-Wesley, 1994.
- [7] *Greedy-Algorithmus*. URL <http://de.wikipedia.org/wiki/Greedy-Algorithmus>. - 18.04.2010
- [8] Ghoffrani, Mehdi: *Development and Intoduction of a flexible software system for configuration of virtual products*. (In German: *Entwicklung und Einführung eines flexiblen Softwaresystems zur Konfigurierung virtueller Produkte*.) Dissertation, Ruhr-Universität Bochum, 2007
- [9] Gronau, N./Gäbler, A.: *Introduction to Business Informatics*. (In German: *Einführung in die Wirtschaftsinformatik {Skripte zur Wirtschaftsinformatik}*). Berlin: GITO-Verlag, 2008.
- [10] Informatik Forum Simon GmbH: *Model Driven Architecture*. URL http://www.infforum.de/themen/anwendungsentwicklung/the_ma_SE_model-driven-architecture.htm. - 04.09.2009
- [11] Harasymowicz-Birnbach, Joanna: *Diagnosis of Effectiveness and Efficiency of Knowledge Management*. (In German: *Effektivitäts- und effizienzorientierte Diagnose des Wissensmanagements*. Bern: 1. Aufl. Haupt Verlag, 2008.
- [12] Kohls, Christian: *Design Patterns to document Expert Knowledge*. (In German: *Design Patterns zur Dokumentation von Erfahrungswissen*.) Tübingen, 2007.
- [13] Lee, Ghang ; Eastman, Charles M. ; Sacks, Rafael: Twelve Design Patterns for Integrating and Normalizing Product Model Schemas. In: *Computer-Aided Civil and Infrastructure Engineering* (2007) Nr. 22, S. 163-181
- [14] Mertins, K. ; Heising, P. ; Finke, I. ; Ulbrich, C.: The Fraunhofer Knowledge Management Audit (FKM-Audit). In: Mertins, K. ; Heising, P. ; Vorbeck, J. (Hrsg.): *Knowlegde Management: Concepts and Best Practices*. Berlin Heidelberg New York : Springer-Verlag, 2003,
- [15] Mertins, K. ; Heising, P. ; Vorbeck, J.: *Knowlegde Management: Concepts and Best Practices*. Berlin Heidelberg New York: 2. Aufl. Springer-Verlag, 2003.
- [16] Petersen, Sönke: Knowledge, Design, Organisation. (In German: Wissen, Design, Organisation.) In: Stary, C. (Hrsg.): *Mensch & Computer 2005: Kunst und Wissenschaft –Grenzüberschreitungen der interaktiven ART*. München : Oldenbourg Verlag, 2005, S. 201-210
- [17] Scheer, Christian: *Customer oriented Product Configurator*. (In German: *Kundenorientierter Produktkonfigurator: Erweiterung des Produktkonfiguratorkonzeptes zur Vermeidung kundeninitiiertter Prozessabbrüche bei Präferenzlosigkeit und Sonderwünschen in der Produktspezifikation*.) Berlin: Logos, 2006.
- [18] Schienmann, Bruno: *Continous Requirement Engineering*. (In German: *Kontinuierliches Anforderungsmanagement: Prozesse, Techniken, Werkzeuge*. München: Addison-Wesley, 2002.
- [19] Sinz, Carsten: *Verification of rule based Configuration Systems*. (In German: *Verifikation regelbasierter Konfigurationssysteme*.) Dissertation, Universität Tübingen, 2003
- [20] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns - Elements of reusable object-oriented Software*. Reading, Mass. [u.a.]: 2. print. Aufl. Addison-Wesley, 1994

AUTHOR INDEX

Albert, Patrick, <i>IBM, France</i>	60
Aldanondo, Michel, <i>Toulouse University, France</i>	21
Astesana, Jean Marc, <i>Renault, France</i>	33
Bazzani, Massimiliano, <i>Universita del Piemonte Orientale, Italy</i>	27
Bossu, Yves, <i>Renault, France</i>	33
Cosserat, Laurent, <i>Renault, France</i>	33
Djefel, Meriem, <i>Toulouse University, France</i>	21
Fabro, Marcos Didonet Del, <i>IBM, France</i>	60
Falkner, Andreas, <i>Siemens AG Österreich, Austria</i>	40
Fargier, Hélène, <i>IRIT, France</i>	33
Felfernig, Alexander, <i>Graz University of Technology, Austria</i>	15
Gaborit, Paul, <i>Toulouse University, France</i>	21
Haag, Albert, <i>SAP AG, Germany</i>	57
Haselböck, Alois, <i>Siemens AG Österreich, Austria</i>	40
Haug, Anders, <i>Technical University of Denmark</i>	9
Hvam, Lars, <i>Technical University of Denmark</i>	9
Küchlin, Wolfgang, <i>Universität Tübingen, Germany</i>	51
Mayer, Wolfgang, <i>University of South Australia</i>	46
Mortensen, Niels Henrik, <i>Technical University of Denmark</i>	9
Pitiot, Paul, <i>Toulouse University, France</i>	21
Plietz, Matthias, <i>ORISA Software GmbH, Germany</i>	63
Portinale, Luigi, <i>Universita del Piemonte Orientale, Italy</i>	27
Schenner, Gottfried, <i>Siemens AG Österreich, Austria</i>	40
Schubert, Monika, <i>Graz University of Technology, Austria</i>	15
Stumptner, Markus, <i>University of South Australia</i>	46
Thiagarajan, Rajesh, <i>University of South Australia</i>	46
Vareilles, Elise, <i>Toulouse University, France</i>	21
Zengler, Christoph, <i>Universität Tübingen, Germany</i>	51