ECAI Patras 2008

18th European Conference on Artificial Intelligence

21-22 July 2008

Workshop Proceedings

Workshop on Configuration Systems

Juha Tiihonen Alexander Felfernig Markus Zanker Tomi Männistö

University of Patras

The 18th European Conference on Artificial Intelligence

Proceedings

Workshop on Configuration Systems

Monday - Tuesday July 21-22, 2008

Patras, Greece

ISBN 978-960-6843-01-3

Juha Tiihonen, Alexander Felfernig, Markus Zanker, Tomi Männistö

Contents

Workshop Organization

Workshop Co-chairs

Juha Tiihonen (Helsinki University of Technology, Finland)
Alexander Felfernig (University Klagenfurt Austria)
Tomi Männistö (Helsinki University of Technology, Finland)
Markus Zanker (University Klagenfurt, Austria)

Organizing Committee

Claire Bagley (Oracle Corporation, USA)
Thorsten Blecker (HITeC c/o University of Hamburg, DE)
Albert Haag (SAP AG, Germany)
Thorsten Krebs (HITeC c/o University of Hamburg, DE)
Barry O'Sullivan (University College Cork, Ireland)
Markus Stumptner (University of South Australia)

Program Committee

Michel Aldanondo (E. d. Mines d'Albi, France)
Tomas Axling (Tacton System AB, Sweden)
Claire Bagley (Oracle Corporation, USA)
Thorsten Blecker (HITeC c/o University of Hamburg, DE)
Alexander Felfernig (University Klagenfurt, Austria)
Felix Frayman (Felix Frayman Consulting, USA)
Gerhard Friedrich (University Klagenfurt, Austria)
Albert Haag (SAP AG, Germany)
Esther Gelle (ABB Switzerland -Power Generation)
Youssef Hamadi (Microsoft Research, UK)
Dietmar Jannach (Technical University Dortmund, DE)
Ulrich Junker (ILOG S.A., France)
Thorsten Krebs (HITeC c/o University of Hamburg, DE)
Diego Magro (Universita di Torino, Italy)
Tomi Männistö (Helsinki University of Technology, Finland)
Klas Orsvarn (Tacton System AB, Sweden)
Barry O'Sullivan (University College Cork, Ireland)
Frank Piller (RWTH Aachen University, Germany)
Marty Plotkin (Oracle Corporation, USA)
Mihaela Sabin (University of New Hampshire, USA)
Carsten Sinz (University of Tuebingen, Germany)
Markus Stumptner (University of South Australia)
Juha Tiihonen (Helsinki University of Technology, Finland)
Paolo Viappiani (University of Toronto, Canada)
Markus Zanker (University Klagenfurt, Austria)

Preface

Configuration problems are among the most fruitful domains for applying and developing AI techniques. Powerful knowledge-representation formalisms are necessary to capture the great variety and complexity of configurable product models. Furthermore, efficient reasoning methods are required to provide intelligent interactive behavior in configuration systems, such as solution search, satisfaction of user preferences, personalization, optimization, diagnosis, etc.

Nowadays, different AI approaches are well-established as central technologies in many industrial configuration systems. This wide-spread industrial use of AI-based configurators makes the field more challenging than ever: the complexity of configurable products still increases, the mass-customization paradigm is extended to fields like service and software configuration, personalized (web-based) user interaction and user preference elicitation are of increasing importance, and finally, the integration of configurators into surrounding IT infrastructures like business information systems or web applications becomes a critical issue.

The main goal of the workshop is to promote high-quality research in all technical areas related to configuration. The workshop continues the series of ten successful Configuration Workshops started at the AAAI'96 Fall Symposium and continued on IJCAI, AAAI, and ECAI since 1999. As such, the workshop is of interest for researchers working in the various fields within the wide range of applicable AI technologies (e.g. Constraint Programming, Description Logics, Non-monotonic Reasoning, Case-Based Reasoning ...). It serves as a platform for researchers and industrial participants to exchange needs, ideas, benchmarks, use cases etc. Beside the participation of researchers from a variety of different fields, past events always attracted significant industrial interest from major configurator vendors like Oracle, SAP, and Tacton, as well as from end-users like ABB, DaimlerChrysler, HP, and Siemens.

Workshop invited talks from industry and academia both have 20 years of experience in configuration. Among technical papers, examples of application domains beyond traditional products include services in form of travel and software configuration. Examples of traditional products included classical car and PC examples, cameras, cranes, elevators, and railway interlocking systems. Personalized recommendation of configurable products or services seems to be emerging and is discussed in two papers. Additional ways of supporting users with richer interaction are presented. Problem solving methods applied in workshop papers include linear inequalities in addition to CSP in traditional, generative, and distributed variants. Other topics include fundamentals of configuration modeling, applying UML for configuration modeling, integrating configuration with production planning, and supporting long-term management with debugging facilities.

The configuration community thrives. Extended versions of this year's best papers are to be published in International Journal of Mass-Customization. In addition, AI EDAM has allocated Spring 2011 issue for a special issue on Configuration. The idea is to form the main body of the special issue from extended versions of best papers from 2009 and 2010 workshops. We hope to attract high-quality submissions to next workshops and the special issue.

Juha Tiihonen, Alexander Felfernig, Tomi Männistö, Markus Zanker

Solving Practical Configuration Problems Using UML

Andreas Falkner[1] and **Ingo Feinerer**[2] and **Gernot Salzer**[2] and **Gottfried Schenner**[1]

Abstract. In the past we have successfully specified configurations of complex systems in the railway domain using fragments of the Unified Modelling Language (UML). Java programs derived from such specifications check constraints and properties of those configurations in a semi-declarative manner.

The aim of this paper is to handle constraints completely declaratively and to use the reasoning component as a black box, avoiding the procedural aspects of Java. We describe the UML elements required for selected practical configuration problems and translate them to inequalities over integers. This approach not only provides a precise semantics but also allows us to perform reasoning tasks (like finding inconsistencies) efficiently. Finally we point out some open problems.

1 INTRODUCTION

Product configuration is the process of creating an individual product based on a description of possible parts and assemblies in a knowledge base or product model. Configuring large-scale systems like railway control systems poses several challenges [11]: They consist of thousands of components and have to satisfy numerous constraints and optimisation criteria. Moreover, such systems are long-lived; configurations have to be maintained and modified over several decades. To be able to handle big configurations automatically it is necessary to employ formal methods, which may lead to the so-called *knowledge acquisition bottleneck*: The development and maintenance of the database requires knowledge engineers which are familiar with the formal representation language *and* the domain.

Using the Unified Modelling Language (UML) for specifying configurations is one particular attempt to tackle this challenge; see e.g. [16]. UML's graphical notations are well-known and widely used in software engineering. Therefore one may expect that the barrier of using them is lower than e.g. for logic-oriented knowledge representation languages, which are complex and familiar to only a few people. Moreover, many tools exist for composing and manipulating UML specifications. Besides, the diagrams can hold additional information about the components such that program code for tasks beyond configuration can be generated from the same specification.

Early in the development of object-oriented modelling it has been already noted that "most object-oriented methods only provide a loose interpretation of the meaning of the diagrams they use. This can lead to problems of: misinterpretation (confusion and disagreement over the precise meaning of a model); analysis (important properties of a model can only be informally verified) and design (correctness of designs cannot be checked)" [10]. The situation improved with the UML standard [18], but much of the criticism still applies. The semantics of UML diagrams are defined in a semi-formal manner, partially resulting in incomplete or ambiguous definitions (see e.g. [17]).

Therefore formal reasoning about UML requires the formalisation of the intended semantics of UML using a rigorous language. Felfernig et al. [15] propose UML class diagrams with is-a relationships for the knowledge acquisition front-end of configuration knowledge bases in the semantic web, and translate them to OIL, a precursor of the ontology language OWL. Other authors translate UML class diagrams to formal languages like Object-Z [19], PVS [20], first-order logic [4], or description logic [5].

Embedding class diagrams into an expressive formal language has the advantage that different formalisms can be translated to the same basic logic and therefore may occur mixed in a specification. For instance, it is possible to express the semantics of constraints written in OCL, the companion of UML, in the same first-order logic. Moreover, well-developed reasoning techniques and theorem provers for these logics can be used to show satisfiability and consistency. E.g., Calvanese et al. show that frame languages, semantic data models and object-oriented data models can be translated to a description logic called $\mathcal{ALUNI}$ and that satisfiability and subsumption of models can be checked in this framework [7].

This flexibility and generality comes at a price, however. Reasoning tasks in expressive logics are of a high computational complexity. E.g., checking the consistency of $\mathcal{ALUNI}$-specifications is EXPTIME-complete [2]. Therefore we decided to extend an approach that was first used by Lenzerini and Nobili for entity-relationship (ER) diagrams [22]. We express the semantics of UML class diagrams by linear inequalities over positive integers. The expressive power is the same as the one of class diagrams, which means that no extra complexity is added. Polynomial graph-based algorithms can be used to check satisfiability and to compute minimal models [9]. The next sections describe this approach in more detail, together with the extensions necessary to cope with UML's uniqueness constraints and lower bounds on the number of objects as first presented in [13, 14].

Section 2 describes UML class diagrams in the context of configuration management. Section 3 gives a compact overview of the translation of class diagrams to inequalities. Sections 4 and 5 present the main contributions: Section 4 describes particular challenges arising from the use of class diagrams for configuration management, whereas section 5 proposes solutions for some of them.

2 UML CLASS DIAGRAMS

To illustrate our approach, let us consider the UML specification in figure 1, which models a technical system with several components: Every section contains at least one and at most twenty elements, each

[1] Siemens AG Österreich, 1030 Vienna, Austria
email: {andreas.a.falkner,gottfried.schenner}@siemens.com
[2] Technische Universität Wien, 1040 Vienna, Austria
email: {feinerer,salzer}@logic.at

with one or two modules. Each module has to be placed in a rack consuming two slots. There are only one or two racks allowed per section, and each of their five slots may be empty (hence a lower cardinality of zero for the modules).

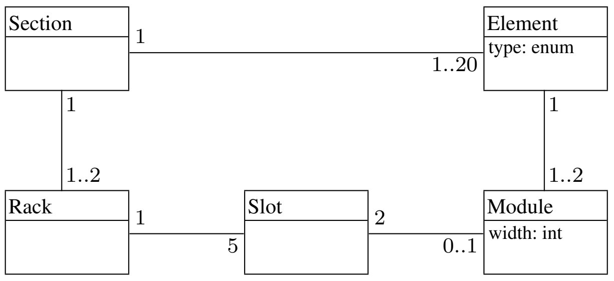


Figure 1. Example of a UML specification

To model such a situation, the following basic elements of UML class diagrams are needed:

Classes represent the types of available components.

Associations define relations between classes or objects. We restrict our attention to binary associations, since general n-ary associations are less relevant in our context.

Multiplicities (also called cardinalities) constrain the number of links between objects.

Association attributes: The ends of associations may be labelled as *unique* (default) or *non-unique* with the meaning that the corresponding multiplicity constrains the number of different objects or the number of links, respectively.

Lower bounds define the minimal number of objects instantiating a particular class in a valid configuration (default 0). Formally, these lower bounds could be defined as simple OCL constraints or by introducing one additional singleton class with a lower-bounded association to each other class.

Methods and attributes of classes (like *type* or *width* in the example) currently have no impact on the computation of valid configurations, but are useful when generating code for the components from the same specification.

Given a specification like in figure 1, several problems are of particular interest:

Consistency: For each class, is there a configuration that satisfies the specification and that instantiates the class with at least one object? Because of specification errors it may happen that the associations are too restrictive and admit no reasonable solution. Suppose we add a new association to our example expressing that each slot is linked to exactly one section, and each section is linked to up to four slots. Then it is not hard to see that the specification can only be satisfied by having no sections, racks, and slots at all: Every section is connected to at least one rack, which is linked to five slots. But according to the new association, each section can be linked to only four of the slots, so another section is needed, which in turn requires at least one rack with five more slots, requiring even more sections. For debugging a specification it is useful to discover such inconsistencies as early as possible.

Construction of valid configurations: Given lower bounds on the number of objects for certain types of components, the task is to complete the configuration in a way such that it satisfies all multiplicities. Ideally, the configuration should be minimal, requiring as few objects of each type as possible. Suppose we get the order to build a section with three elements, i.e., the lower bound is one for sections and three for elements. Then a configuration satisfying the specification in figure 1 also needs at least three modules, six slots, and two racks.

Our approach to handle these problems is to transform the associations to equivalent inequalities. For instance, the relationship between the two classes *Slot* and *Module* in figure 1 can be expressed by the inequalities

$$0 \cdot |\mathit{Slot}| \leq 2 \cdot |\mathit{Module}|$$
$$2 \cdot |\mathit{Module}| \leq 1 \cdot |\mathit{Slot}|$$

where $|\mathit{Slot}|$ and $|\mathit{Module}|$ denote the required numbers of slots and modules, respectively. These inequalities are equivalent to the diagram in the sense that every solution corresponds to an object model of the class diagram, and every model of the class diagram corresponds to a solution of the inequalities. The next section describes this transformation in more detail.

3 FROM UML TO LINEAR INEQUALITIES

Consider the general situation depicted in figure 2, where A and B are two classes related by an association with multiplicities $m_1..m_2$ and $n_1..n_2$, respectively. Each of the attributes u and v may have the value *unique* or *non-unique*. The situation $u =$ unique means that every A-object is linked to at least m_1 and at most m_2 *different* B*-objects*, whereas $u =$ non-unique means that every A-object has at least m_1 and at most m_2 *links* to B-objects, which do not have to be different from each other. The attribute v is interpreted analogously, with the roles of the classes A and B reversed.

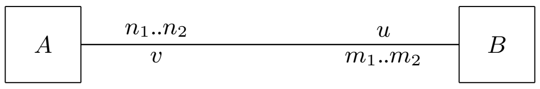


Figure 2. Binary association with multiplicities – the general setting

The inequalities characterising the association depend on the values of u and v. We distinguish three cases.

$\boldsymbol{u = v =}$ **non-unique:** The numbers of objects instantiating the classes A and B, denoted by $|A|$ and $|B|$, satisfy the inequalities

$$m_1 \cdot |A| \leq n_2 \cdot |B|$$
$$n_1 \cdot |B| \leq m_2 \cdot |A|$$

The number ℓ of links between objects of class A and class B may take any value between the following bounds:

$$\max(m_1 \cdot |A|, n_1 \cdot |B|) \leq \ell \leq \min(m_2 \cdot |A|, n_2 \cdot |B|)$$

$\boldsymbol{u = v =}$ **unique:** The inequalities are the same as above, plus two more:

$$|A| > 0 \implies |B| \geq m_1$$
$$|B| > 0 \implies |A| \geq n_1$$

When counting links only once for every pair of objects, their number can be bounded by the following expressions:

$$\max(m_1 \cdot |A|, n_1 \cdot |B|) \leq \ell \leq \min(m_2 \cdot |A|, n_2 \cdot |B|, |A| \cdot |B|)$$

u = non-unique and v = unique: This case is a combination of the other two plus an additional inequality.

$$\begin{aligned} & n_1 \cdot |B| \leq m_2 \cdot |A| \\ m_1 > 0 \implies & |A| \leq n_2 \cdot |B| \\ |B| > 0 \implies & |A| \geq n_1 \end{aligned}$$

For the number of links we obtain a new upper bound:

$$\max(m_1 \cdot |A|, n_1 \cdot |B|) \leq \ell \leq \min(m_2 \cdot |A|, m_2 \cdot n_2 \cdot |B|)$$

Note that for $m_1 > 0$ the upper bound simplifies to $m_2 \cdot |A|$.

The fourth case, u = unique and v = non-unique, is symmetric to this one.

As an example, consider the binary association in figure 3. We specify the requirement that we want to have at least one B-object directly in the box of B, instead of adding an OCL constraint. The

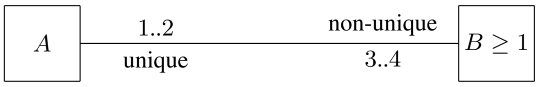


Figure 3. Binary association with multiplicities – an example

corresponding inequalities, one specifying the lower bound and three for the unique/non-unique case, have the form

$$\begin{aligned} & |B| \geq 1 \\ & 1 \cdot |B| \leq 4 \cdot |A| \\ 3 > 0 \implies & |A| \leq 2 \cdot |B| \\ |B| > 0 \implies & |A| \geq 1 \end{aligned}$$

The minimal solution of these inequalities is $|A| = |B| = 1$; the number ℓ of links is bounded by $3 \leq \ell \leq 4$. It might not be completely intuitive that the minimal configuration (figure 4) consists of only one object for each class (in spite of multiplicity 3..4), but three or four links (in spite of multiplicity 1..2). The point is of course that the first multiplicity constrains *links* and the second one *objects*.

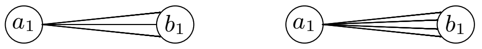


Figure 4. Minimal models for the specification in figure 3 with $\ell = 3$ and $\ell = 4$

In [13, 14], the following results are proved:

- The class diagram is consistent if and only if the inequalities admit a non-trivial solution, i.e., if at least one variable is assigned a value different from zero.
- The inequalities are complete, i.e., every valid object model of the class diagram satisfies the corresponding inequalities.
- The inequalities are correct, i.e., every solution for the cardinalities and for the number of links corresponds to a valid object model of the diagram.
- The solutions are closed under linear combination and minimum operation. This implies that the minimal solution (requiring the least number of objects) is uniquely determined. Moreover, if there is a non-trivial solution, then there are infinitely many ones.
- The inequalities can be solved efficiently over the rational numbers using shortest-path algorithms for graphs. The rational solution immediately gives an integer solution. The minimal solution is harder to find, but usually requires only little extra effort.
- General n-ary associations can be handled similarly, as long as all n ends of an association carry the same label, unique or non-unique.

4 CHALLENGES

The approach described in the last section handles many interesting situations correctly. However, in applications where the product is a large system with many components like a railway interlocking system, additional aspects are important [12]. To illustrate the issues consider the specification in figure 1, which is a simplified fragment of a real-world specification in that domain.

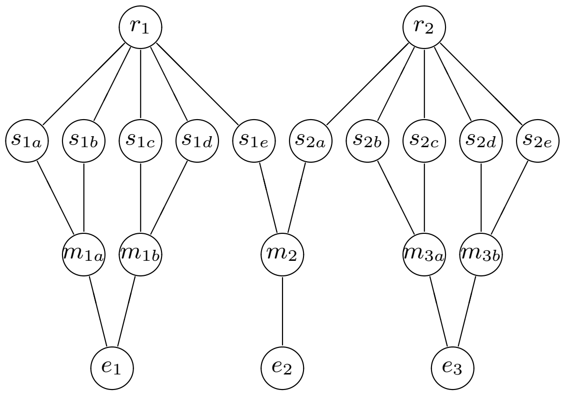


Figure 5. Configuration satisfying the specification in figure 1 (r = racks, s = slots, m = modules, e = elements, section omitted)

The following constraints are not captured by the UML model:

1. *"All slots occupied by a module must belong to the same rack,"* and *"all modules of an element must be placed in the same rack."* According to the semantics of UML class diagrams, figure 5 shows a valid configuration (we omit the section, which is connected to all racks and elements). However, it violates the additional constraint, since module m_2 is connected to slots of different racks. To satisfy the constraint we need a third rack which in turn requires a second section, or we must remove one module of e_1 or e_3.
2. *"The rack housing an element – via the slots and modules – must belong to the same section as the element."* UML class diagrams without OCL do not offer any means to specify that the relation Section-Element should be a subrelation of the composed relation Section-Rack-Slot-Module-Element, or that these two relations should be equal.
3. *"The objects of class 'Element' have an attribute 'type' with value '1-module' or '2-module' which controls the number of modules connected to the element."* According to the specification we have the choice to connect each element to either one or two modules. The intention, however, is to have two types of elements competing for the same slots. How many elements of each type are to be used is not up to the configuration program but usually determined in advance. Therefore, removing a module from Fig. 5 as mentioned above in challenge 1 is no viable option.

Moreover, there are tasks beyond checking for consistency or minimising configurations with respect to the number of objects that are useful in practice.

4. *Reasoning about cardinalities.* In figure 1 the maximum of twenty elements per sections can never be reached. A section has at most two racks, therefore at most ten slots, therefore at most five modules, and therefore at most five elements. If the additional constraint requiring the same rack for the modules of an element is taken into account, it can be at most four elements. This is a static statement about the model. The discrepancy between four and twenty may indicate a specification error, like inconsistency does.
5. *Partial configurations.* Similar statements can be derived for partial configurations: If a section contains already an element with two modules, then there are only two (not three) additional elements allowed. This is a relevant information for the user.
6. *Immediate feedback.* Getting this kind of information fast is vital for a staged configuration process. The user creates sections and elements first, based upon the logics of the system. The (intermediate) result might already be used for answering a tender. The configuration of the hardware is a second step, e.g. after winning the bid. However, the user must get the information whether his/her setting of sections and elements will yield a correct hardware configuration as early as possible, so that there are no unexpected costs for additional sections. Even if the hardware configuration itself is done before the bid, it is important to know whether there is a complete solution. Generating the solution itself (i.e. placement of the modules) is complicated and time-consuming and cannot be done on-the-fly. Furthermore it will take longest if there is no valid solution (similar to pigeon-hole problems).

The next section describes extensions of our approach that address some of the challenges.

5 SOME SOLUTIONS

5.1 The attribute 'all-same'

One recurring requirement is that the links of an object must not be connected to several objects instantiating the partner class, but that they all have to connect to the same object like stated in the first challenge in section 4. To be able to model such constraints we propose a new attribute that we call 'all-same'. It has essentially the same meaning as the attribute 'non-unique', i.e., the corresponding multiplicity constrains the links, but it additionally requires that all links of an object have to end at the same partner object.

The new attribute solves some of the issues described in the last section. For instance, if we avoid to model the slots of a rack by a separate class (figure 6) then 'all-same' ensures that every module is associated with exactly one rack.

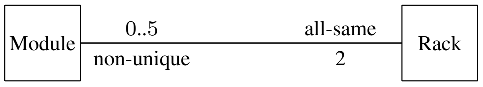


Figure 6. Example using the all-same attribute

We investigate now the interaction of the new attribute with the other two, i.e., we have to consider three cases. For the general setup see figure 2.

u = all-same and v = non-unique: The number of objects instantiating the classes A and B satisfy the inequalities:[3]

$$\begin{aligned} m_2 > 0 &\implies \lceil \tfrac{n_1}{m_2} \rceil \cdot |B| \le |A| \\ m_1 > 0 &\implies \qquad |A| \le \lfloor \tfrac{n_2}{m_1} \rfloor \cdot |B| \end{aligned}$$

The number of links may take values within the following bounds:

$$\begin{aligned} \ell &\ge \max(m_1 \cdot |A|, n_1 \cdot |B|) \\ \ell &\le |A| \bmod |B| \cdot \min(m_2 \cdot \lceil \tfrac{|A|}{|B|} \rceil, n_2) \\ &\quad + (|B| - |A| \bmod |B|) \cdot \min(m_2 \cdot \lfloor \tfrac{|A|}{|B|} \rfloor, n_2) \end{aligned}$$

The upper bound simplifies to $m_2 \cdot |A|$ for $m_2 \cdot \lceil \frac{|A|}{|B|} \rceil \le n_2$, and to $n_2 \cdot |B|$ for $n_2 \le m_2 \cdot \lfloor \frac{|A|}{|B|} \rfloor$.

As an example, the number of objects in figure 6 are constrained by

$$\begin{aligned} 2 > 0 &\implies \lceil \tfrac{0}{2} \rceil \cdot |\mathit{Rack}| \le |\mathit{Module}| \\ 2 > 0 &\implies \quad |\mathit{Module}| \le \lfloor \tfrac{5}{2} \rfloor \cdot |\mathit{Rack}| \end{aligned}$$

which is equivalent to $|\mathit{Module}| \le 2 \cdot |\mathit{Rack}|$. This constraint correctly reflects the semantics of the 'all-same' attribute: Since both links of each module have to connect to the same rack, at most four of the five slots of the rack can be utilised, i.e., the number of modules is at most twice the number of racks. The number of racks is unbounded, since the cardinality $0..5$ admits empty racks not connected to any module. For the links we obtain

$$\begin{aligned} &\ell \ge \max(2 \cdot |\mathit{Module}|, 0 \cdot |\mathit{Rack}|) = 2 \cdot |\mathit{Module}| \\ &\ell \le 2 \cdot |\mathit{Module}| \quad (\text{since } 2 \cdot \lceil \tfrac{|\mathit{Module}|}{|\mathit{Rack}|} \rceil \le 2 \cdot \lceil \tfrac{2 \cdot |\mathit{Rack}|}{|\mathit{Rack}|} \rceil = 2 \cdot 2 \le 5) \end{aligned}$$

i.e., we have $\ell = 2 \cdot |\mathit{Module}|$. This is intuitively correct since each module requires exactly two links.

u = all-same and v = unique: For such an association the cardinalities of the classes satisfy the following inequalities:

$$\begin{aligned} & n_1 \cdot |B| \le |A| \\ m_1 > 0 \implies & |A| \le n_2 \cdot |B| \end{aligned}$$

The number of links is bounded by:

$$\max(m_1 \cdot |A|, n_1 \cdot |B|) \le \ell \le \min(m_2 \cdot |A|, m_2 \cdot n_2 \cdot |B|)$$

which is the same interval as in the case non-unique/unique. Note that for $m_1 > 0$ the inequalities can be more compactly written as $n_1 \cdot |B| \le |A| \le n_2 \cdot |B|$ and $m_1 \cdot |A| \le \ell \le m_2 \cdot |A|$.

$u = v$ = all-same: Such an association admits non-trivial solutions only for $m_1 \le n_2$ and $n_1 \le m_2$, i.e., the two multiplicities have to overlap. In this case the cardinalities $|A|$ and $|B|$ are related by the following inequalities:

$$\begin{aligned} m_1 > 0 &\implies |A| \le |B| \\ n_1 > 0 &\implies |B| \le |A| \end{aligned}$$

The number of links may take values within the following bounds:

$$\min(|A|, |B|) \cdot \max(m_1, n_1) \le \ell \le \min(|A|, |B|) \cdot \min(m_2, n_2)$$

[3] $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x ('floor' of x), $\lceil x \rceil$ the least integer greater than or equal to x ('ceiling' of x).

Proposition 1 *The inequalities above are complete and correct, i.e.: Given a binary association with multiplicities and at least one attribute 'all-same', then the numbers of objects and links in every instantiation of the association satisfy the inequalities, and conversely, every solution of the inequalities corresponds to an instantiation with the appropriate number of objects and links.*

Note that the statements at the end of section 3 concerning the solvability of inequalities and the properties of the solutions also apply to the inequalities in this section, since the inequalities here are of the same form as those there.

5.2 Redundant cardinalities

Challenge 4 in section 4 requires reasoning about cardinalities. A limited form of such reasoning is obtained as a by-product of the algorithm for checking the consistency of inequalities [14], which computes the strongest relationship between any two classes. For the inequalities corresponding to the specification in figure 1, this algorithm derives the inequality $1 \cdot |\mathit{Element}| \leq 5 \cdot |\mathit{Section}|$ from

$$\begin{aligned} 1 \cdot |\mathit{Element}| &\leq 1 \cdot |\mathit{Module}| \\ 2 \cdot |\mathit{Module}| &\leq 1 \cdot |\mathit{Slot}| \\ 1 \cdot |\mathit{Slot}| &\leq 5 \cdot |\mathit{Rack}| \\ 1 \cdot |\mathit{Rack}| &\leq 2 \cdot |\mathit{Section}| \end{aligned}$$

The new inequality makes the inequality $1 \cdot |\mathit{Element}| \leq 20 \cdot |\mathit{Section}|$, which corresponds to the direct association between sections and elements, redundant. Hence the algorithm does not only check for consistency but also provides information about cardinalities that cannot be fully exploited.

5.3 Subclassing

Challenge 3 in section 4 addresses situations where instances of the same class should behave differently in configurations depending on some property, like the objects of class 'Element' in figure 1, where the attribute 'type' distinguishes several types of elements requiring different numbers of modules. A more UML-ish way to model such a situation is by using subclasses. Every value of the type defines a separate subclass, each with an association to class 'Module' specialising the general association between the classes 'Element' and 'Module' (figure 7). The interaction of subclass hierarchies with cardinalities has been already investigated in the context of ER-diagrams [6], where the authors describe a method to eliminate hierarchies by introducing additional classes. This reduction is fairly complex and may be exponential in the worst case. A recent paper [3] proposes a much simpler algorithm that is able to cope with class diagrams and generalisation set constraints as defined in UML 2.0.

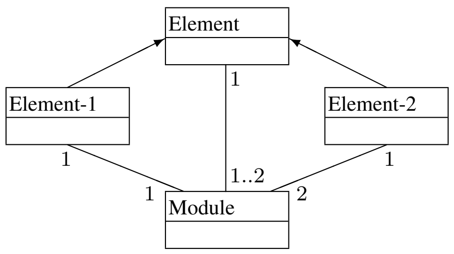


Figure 7. Different types of elements modelled as subclasses

Even though our configuration problems need hierarchies, we use only a small part of the possibilities offered by UML. Therefore our aim is to tailor the general approach to our particular needs to become as efficient as possible. For instance, the specification above can be translated to a single equation:

$$1 \cdot |\mathit{Element\text{-}1}| + 2 \cdot |\mathit{Element\text{-}2}| = 1 \cdot |\mathit{Module}|$$

Our types of hierarchies translate to linear equalities and inequalities, possessing in general more than two variables. Integer programming with two variables is already NP-complete [21][4], and adding variables increases complexity even further. However, this additional complexity is intrinsic to the original problem and thus cannot be helped; it is not caused by modelling the configuration problem by numerical (in)equalities. Moreover, linear equalities and inequalities over integers occur in many fields like integer programming or unification theory, and several efficient algorithms for solving them have been proposed [1, 8].

5.4 Generate and test

Our approach of translating specifications of configuration problems to linear inequalities is fairly general, but of course has its limitations. In particular, the implicit assumption so far was that the placement of links is only constrained by the cardinalities. If additional constraints rule out certain links, then we may need more objects than suggested by the inequalities. If, for instance, objects of subtype 'Element-2' (see figure 7) are preferable costwise to those of subtype 'Element-1', we might accept more racks than necessary in an object-minimal configuration. Numeric cost functions might still be compatible with our representation of class diagrams. Other constraints like mutual exclusion conditions, however, can hardly be handled by our numeric framework. In such cases we can still use generate-and-test as a fallback solution: The algorithm for solving inequalities generates presolutions that satisfy most of the constraints, and in a second step these solutions are tested with respect to the remaining constraints. Even if we have to iterate several times between these two stages, the overall procedure stays usable as long as the first stage is fast, which it is, and the number of extra constraints to be handled by the second stage is small. These considerations provide at least a partial answer to challenge 6 of section 4.

6 CONCLUSION

This paper presented an approach that promises to deal efficiently and declaratively with practical configuration problems. After summarising the basic translation of class diagrams to inequalities we described specific challenges that in our experience are of particular importance. We proposed solutions for some of them. Most notably, we introduced the new attribute 'all-same', we argued that checking for consistency also provides information about redundant cardinalities, and we presented ideas how to deal with subclasses in our framework.

Several open problems remain. On the theoretical side, handling composed associations and their relationship to other associations (challenge 2) and using the approach not only for static analysis

[4] This is no contradiction to the fact that we are able to solve the inequalities encountered so far efficiently. The reason is that we need no upper bounds, i.e., we do not have inequalities of the form $ax + by \leq c$ where a, b, c are constants and x, y are variables.

(challenge 5) are the most important ones. On the practical side we think that the theory is sufficiently developed by now such that it is worth the effort to integrate the algorithms into an integrated development environment that is capable of manipulating UML specifications, like Eclipse. Such an implementation will allow us to test our approach by applying it to configuration problems that are comparable in structure, size, and complexity to real-world problems.

REFERENCES

[1] Farid Ajili and Evelyne Contejean, 'Avoiding slack variables in the solving of linear diophantine equations and inequations', *Theoretical Computer Science*, **173**(1), 183–208, (1997).

[2] *The Description Logic Handbook: Theory, Implementation, and Applications*, eds., Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, Cambridge University Press, 2003.

[3] Mira Balaban and Azzam Maraee, 'Consistency of UML class diagrams with hierarchy constraints', in *Next Generation Information Technologies and Systems*, volume 4032 of *Lecture Notes in Computer Science*, pp. 71–82. Springer Berlin / Heidelberg, (2006).

[4] Bernhard Beckert, Uwe Keller, and Peter Schmitt, 'Translating the Object Constraint Language into First-order Predicate Logic', in *Proceedings, VERIFY, Workshop at Federated Logic Conferences (FLoC), Copenhagen, Denmark*, (2002).

[5] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo, 'Reasoning on UML class diagrams', *Artificial Intelligence*, **168**(1–2), 70–118, (2005).

[6] Diego Calvanese and Maurizio Lenzerini, 'On the interaction between ISA and cardinality constraints', in *Proceedings of the Tenth International Conference on Data Engineering*, pp. 204–213, Washington, DC, USA, (1994). IEEE Computer Society.

[7] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi, 'Unifying class-based representation formalisms', *Journal of Artificial Intelligence Research*, **11**, 199–240, (1999).

[8] Evelyne Contejean and Hervé Devie, 'An efficient incremental algorithm for solving systems of linear diophantine equations', *Information and Computation*, **113**(1), 143–172, (1994).

[9] Konrad Engel and Sven Hartmann, 'Constructing realizers of semantic entity relationship schemes', Technical report, Universität Rostock, Fachbereich Mathematik, 18051 Rostock, Germany, (1995).

[10] Andy S. Evans, 'Reasoning with UML class diagrams', in *WIFT '98: Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques*, p. 102, Washington, DC, USA, (1998). IEEE Computer Society.

[11] A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, 'Benefits from three configurator generations', in *Innovative Processes and Products for Mass Customization*, eds., Th. Blecker, K. Edwards, G. Friedrich, L. Hvam, and F. Salvodor, volume 3, pp. 89–103, Berlin, (2007). GITO-Verlag.

[12] Andreas Falkner and Gerhard Fleischanderl, 'Configuration requirements from railway interlocking stations', in *IJCAI-01 Workshop on Configuration*, Seattle, WA, (August 2001).

[13] Ingo Feinerer, *A Formal Treatment of UML Class Diagrams as an Efficient Method for Configuration Management*, Dissertation, Theory and Logic Group, Institute of Computer Languages, Vienna University of Technology, Austria, March 2007.

[14] Ingo Feinerer and Gernot Salzer, 'Consistency and minimality of UML class specifications with multiplicities and uniqueness constraints', in *Proceedings of the 1st IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering, June 6–8, 2007, Shanghai, China*, pp. 411–420. IEEE Computer Society Press, (2007).

[15] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker, 'Configuration knowledge representations for semantic web applications', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 31–50, (2003).

[16] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker, 'Configuration knowledge representation using UML/OCL', in *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pp. 49–62, London, UK, (2002). Springer-Verlag.

[17] Gonzalo Génova, Juan Llorens, and Paloma Martínez, 'The meaning of multiplicity of n-ary associations in UML', *Software and System Modeling*, **1**(2), 86–97, (2002).

[18] Object Management Group, *Unified Modeling Language 2.1.1 Superstructure Specification*, 2007. `http://www.uml.org/#UML2.0`.

[19] Soon-Kyeong Kim and David A. Carrington, 'Formalizing the UML class diagram using Object-Z', in *UML'99: The Unified Modeling Language - Beyond the Standard, Second International Conference, Fort Collins, CO, USA, October 28-30, 1999, Proceedings*, eds., Robert B. France and Bernhard Rumpe, volume 1723 of *LNCS*, pp. 83–98. Springer, (1999).

[20] P. Krishnan, 'Consistency checks for UML', in *APSEC '00: Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, p. 162, Washington, DC, USA, (2000). IEEE Computer Society.

[21] Jeffrey Lagarias, 'The computational complexity of simultaneous diophantine approximation problems', *SIAM J. on Computing*, **14**(1), 196–209, (1985).

[22] Maurizio Lenzerini and Paolo Nobili, 'On the satisfiability of dependency constraints in entity-relationship schemata', *Information Systems*, **15**(4), 453–461, (1990).

Modeling, Representing, and Configuring Restricted Part-Whole Relations

Lothar Hotz [1]

Abstract. Part-whole relations are the backbone of configuration systems. In this paper, part-whole relations are combined with other relations and restrictions leading to here-called *restricted aggregates*. Depending on what is given, aggregates, parts, or/and relations between parts, different tasks have to be solved. General aggregation reasoning chunks are developed and represented with a configuration language. The approach is applied to the domain of constructing scene interpretations.

1 Introduction

Configuration is the task of composing parameterisable objects (*parts*) to wholes (*aggregates*) such that a given goal is fulfilled by the resulting construction. For this task, descriptions of aggregates and parts of a domain (*domain objects*) are given in a configuration model and a configuration tool is used to create a certain construction. An aggregate is typically described by its potential parts and further restrictions that have to be fulfilled between the parts. Thus, those *restricted* aggregates require that certain relations and predicates are true for their parts. Domain objects have parameters (i.e. relations to primitive data types) and n-ary relations to other domain objects, which can be given by a *task specification* or can be computed by the configuration tool (parameters are set and relations are established).

Configuration technologies are applied in technical areas like automotive, telecommunication, but also in software, services and construction of scene interpretations.

In the following, we develop general aggregation reasoning chunks that allow to construct aggregates from given parts and integrate parts in given aggregates. Furthermore, restricted aggregates, as they are considered here, require certain relations and predicates to be true for their parts. Also, when given such relations between parts, appropriate aggregates should be created that can contain those restricted parts. Thus, for diverse, here called, *key situations* (parts given with or without relations, aggregates given with or without parts) the general aggregation reasoning chunks will construct complete aggregates that guarantee the validity of the restriction.

As an example, we map the general aggregation reasoning chunks to a structure-based configuration language, such that the aggregate can be computed by an appropriate configuration tool [10]. Other representations e.g. with rule languages or usual programming languages are conceivable [11].

The general aggregation reasoning chunks are applied in the domain of Scene Interpretation, which has already been shown to be a configuration task [9]. In this domain, aggregates like *entrance* or *balcony* consist of certain parts like *door*, *railing*, *stairs*, *sign*, *window*. Parts have to fulfill restrictions like spatial relations of the kind *stairs below door*, *sign left-neighbor door* etc. An aggregate may additionally occur as part of a comprising aggregate like *facade*. The task is to construct a description with parts and aggregates of a scene given an image.[2]

In other approaches, compositional relations between parts and aggregates are considered from the epistemological point of view [19, 21, 15]. Beside technical aspects of how to represent parts and wholes with a given configuration language, the relationship between the compositional relation and further restrictions are less considered in those contributions. In this paper, the interplay between those relations are analyzed.

In principle, configuration technologies as described in [2, 4, 13, 17, 7, 12] provide a basic framework for constructing restricted aggregates. While this includes means for modeling, representing, and processing part-whole relations, it is seldom clarified how these facilities are *applied* or *used* for creating part-whole relations, or especially restricted part-whole relations.

This paper tries to further close this gap. Thus, we provide an analysis of possible aggregation situations and their representations by means of concepts and constraints. For this task, we first prescribe the *structure-oriented configuration approach*, which is applied here (Section 2), than consider the problem in more detail (Section 3). Section 4 describes the general solution of our approach for creating restricted part-whole relations and an implementation with a configuration language. Sections 5 and 6 describe some experiments and give a summary of the approach, respectively.

2 Structure-oriented Configuration Approach

As background, we follow the structure-oriented configuration approach as it is described by [1, 2, 3, 4, 5, 8, 16, 17, 18, 22]. While this approach has several variants we focus in the following on four separate knowledge types which can be used for modeling a certain domain:

Concept Hierarchy Domain objects are described using a highly expressive description language providing *concepts*, a taxonomical hierarchy (based on the `is-a` relation), and structural relations like the compositional hierarchy based on the `has-parts` relation. Parameters specify domain-object attributes with value intervals, sets of values (enumerations), or constant values. *Instances* selected for a concrete construction are instantiations of the concepts and represent concrete domain objects. Parameters

[1] HITeC e.V., University of Hamburg, Germany, email: hotz@informatik.uni-hamburg.de

[2] Our task differs to [20] in the fact that we construct descriptions of scenes (including aggregates and parts), while [20] computes spatial arrangements of non-aggregated scene objects.

and structural relations of a concept are also referred to as *properties* of the concept. When instantiated, the properties of an instance are initialized by the values or value ranges specified in the concepts.

Constraints *Constraints* pertaining to properties of more than one object are administered by a constraint net. *Conceptual constraints* are formulated as part of the configuration model. Conceptual constraints consists of a condition and an action part. The condition specifies a *structural situation* of instantiated concepts. If this structural situation is fulfilled by some instances, *constraint actions* that are formulated in the action part are instantiated to *constraints*. Constraint actions can represent restrictions between properties (i.e. *constraint relations*) or operations like `create-instance` (i.e. *constraint operations*). Constraints can be multi-directional, i.e. propagated regardless of the order in which constraint variables are instantiated or changed. At any given time, the remaining possible values of a constraint variable are given as intervals, value sets, or constant values.

Task Description A configuration task is specified in terms of an aggregate which must be configured (the goal) and possibly additional restrictions such as choices of parts, prescribed properties, etc. Typically, the goal is represented by the root node of the compositional hierarchy.

Procedural Knowledge The configuration process provides a stepwise composition of a construction. Each step is one of the following kinds of construction steps: *aggregate instantiation* (or *top-down structuring*), *part integration* (or *bottom-up structuring*), *instance specialization*, and *parameterization*. A step reduces a property value of an instance to a subset (*reduced value*) or finally to a constant (*fixed value*).
After each step the constraint net is optionally propagated. Configuration strategies are used to organize this configuration process in a declarative manner. For example, it is possible to prescribe phases of bottom-up or top-down processing conditioned on certain features of the evolving construction.

For every of the above mentioned knowledge types, a specific language is given which allows to express domain objects with their attributes and restrictions as well as the configuration process (see e.g. the Configuration Knowledge Modeling Language CKML described in [10]). In Figure 1, we sketch some constraint actions of the constraint language part of CKML that are needed for the following examples. The constraint operation `create-instance` and `ensure-relation` are the main mechanism for creating instances and relations between them. `create-instance` instantiates a concept after selecting one of given concept types.[3] Before establishing a relation between given instances, `ensure-relation` checks whether the relation already exists. Numeric constraint relations are used for comparing and computing parameter values given by constants, intervals, or enumerations in the typical mathematical form (which includes interval arithmetic).

For representing aggregates with their parts, spatial relations and restrictions, i.e. for representing *restricted aggregates*, concepts with their taxonomical and compositional relations as well as constraints can be applied, leading to *restricted aggregate models*. However, using this configuration technology for scene interpretation, the question arises, how these facilities can effectively be used both for modelling a domain (i.e. the analysis and in-depth understanding of the domain) and operationalizing this knowledge for the scene interpretation process.

Configuration systems considered here typically do not reason about concepts, but only about instances (for other approaches see e.g. [14]). Thus, the language facilities provided by such systems use instances for reasoning, e.g. conceptual constraints are fulfilled if an instance relation structure matches the structural situations. Thus, instead of description logics, which may also reason about concepts, in configuration systems, appropriate instances have to be created and for the created instances knowledge entities have to be inserted in the configuration model.[4] The question now is, what knowledge entities have to be inserted for reasoning about restricted aggregates?

create-instance	Creates a new instance for one of given concept types.
ensure-relation	Establishes a relation of of a given name between two instances.
less, greater, equal ...	Numeric constraint relations.

Figure 1. Predefined operations and relations (i.e. constraint actions) that are used in the following.

3 Requirements and Application Example

In the following, we consider aggregates locally described by their parts and restrictions. With a *local representation* of aggregates, all information about their parts are kept at one place, i.e. are not distributed over the configuration model. Especially restrictions between parts of different aggregates are not allowed. However, a part may be part of different alternative aggregates, e.g. a door might be part of an entrance or a balcony. Examples of locally defined aggregates are illustrated in Figure 2. It shows a general `Scene-Aggregate` and a specific `Entrance` aggregate with parts and several spatial relations between them. For the aggregate and parts, concept types (e.g. `Stairs`, `Door`), parameters (e.g. `size-x`), and relations between the parts (e.g. `belowNeighbor`, `overlap`) can be described. In the example, we consider spatial relations of parts, however, arbitrary n-ary restrictions between properties of the aggregate's parts can be specified with aggregate restrictions.

Beside the typical signature for concept types, parameters and relations, discriminators are given which describe a *sufficient condition* for the aggregate - if the discriminator holds, then the aggregate should exist. A conjunctive combination of parts and relations can be given as a discriminator. Thus, a discriminator can be only one part or several parts, as well as one or more relation between parts. In the example in Figure 2, six discriminators are given each consisting of one relation (e.g. `?b-s-c` representing `?stairs0 below ?canopy4`).

Each discriminator is a "unique selling point" of an aggregate in the sense that it distinguishes this aggregate from other aggregates. However, once an aggregate has been created, also other restrictions are examined, e.g. further required parts. These restrictions are *necessary conditions*. With this mechanism, complete descriptions of aggregates are created. Note that this may lead to hypotheses of parts

[3] Concept types can be selected e.g. by considering probabilities. In our current application domain, statistics can be computed from annotated images. However, this aspect is not further discussed in this paper.

[4] However, reasoning about concepts is restricted (e.g. n-ary roles are not provided in description logic systems) and thus, less used for configuration tasks.

which may exist or have to exist, if the aggregate exists. For example in the facade domain, a *door* of an *entrance* is hypothesized, if the above mentioned condition `?stairs0 below ?canopy4` is fulfilled through observed *stairs* and *canopy*.[5]

This reasoning is embedded in a backtracking environment as typically provided by configuration systems. Backtracking occurs when inferred decisions lead to a conflict, e.g. signified by the constraint net. In this case, conflict resolution mechanisms should be applied [10].

In the following, we assume that such aggregate models are manually created or learned from a set of given cases e.g. by Version Space Learning, see [6]. Especially discriminators (i.e. sufficient conditions) have to be identified by those methods.

Requirements for the general aggregation reasoning chunks. The aggregation chunks which we target should cover diverse *key situations*, which are given through the task specification or may come up through reasoning during the configuration process. The key situations are described in terms of given instances and properties. A property is *given*, if its value is fixed or reduced (see Section 2), otherwise it is *original* as it was specified in the concept (e.g. a parameter `pos-x-1` is `[0 inf]`). The chunks should:

- allow the construction of a new aggregate when one or more parts with relations or parameters are given (*bottom-up structuring*),
- allow the construction of new parts when an aggregate is given (*top-down structuring*),
- integrate given parts in given aggregates (*bottom-up integration*),
- use given parts for decomposing given aggregates (*top-down decomposition*),
- check given restrictions when all or some parts and aggregates are present (*aggregate consistency*),
- establish the described restrictions when parts and aggregates do not yet fulfill the restrictions (*restriction establishment*),
- determine appropriate types of aggregates and parts when certain relations but no specific type information are given (*object specialization*),
- select one aggregate types and parts when several alternatives are possible (*type selection*).

Modeling of restrictions. For modeling restrictions, we consider a two step approach: First, *physical parameters* are obtained through external systems, e.g. image processing systems, that supply geometric parameters like position and size of objects. These parameters cover *implicit relations* between objects, in particular *spatial predicates* as defined by predicates such as `above-p`. Those relations can be made explicit by establishing appropriate *explicit relations* between objects, such as `above`, `left`, `right` etc. Explicit relations are mainly used for describing restrictions on a high level as illustrated in Figure 2. They abstract from concrete numbers representing physical parameters.

Depending on the key situation, the relations are computed, e.g.:

- If an explicit relation between objects is given, the physical parameters should be changed, so that the geometry holds between the parameters. For example, in a two dimensional x/y coordination system, if an object $o1$ is identified to be below $o2$, the position parameter y of $o1$ should be higher than y of $o2$ (expecting the origin to be at the top left corner).
- If an implicit relation between physical parameters is given, the explicit relation between objects should be established. For example, in a two dimensional x/y coordination system, if the position parameters of $o1$ (i.e. y) is higher than y of $o2$, the relation $o1$ is $below$ $o2$ should be established.
- If the physical parameters are changed, the explicit spatial relations should be checked, if they hold.

```
(define-aggregate :name Scene-Aggregate
  :parameters
  ((size-x [0 inf])
   (size-Y [0 inf])
   (pos-x-1 [0 inf]) (pos-y-1 [0 inf])
   (pos-x-2 [0 inf]) (pos-y-1 [0 inf])
   (parts-top-left-x-variability [0 inf])
   (parts-top-left-y-variability [0 inf])
   (parts-bottom-right-x-variability [0 inf])
   (parts-bottom-right-y-variability [0 inf]))
  :parts
  ((:name ?parts :type Scene-Aggregate
               :number-restriction [0 inf]))
  :restrictions
   ((:name ?variability
     :constraint
       (check-variability ?a ?parts))
    (:name ?bounding-box
     :constraint
       (check-bounding-box ?a ?parts))))

(define-aggregate :name Entrance
  :super Scene-Aggregate
  :parameters
  ((size-x [184 295])
   (size-Y [299 420])
   (parts-top-left-x-variability [7 131])
   (parts-top-left-y-variability [1 284])
   (parts-bottom-right-x-variability [7 131])
   (parts-bottom-right-y-variability [1 284]))
  :parts
  ((:name ?stairs0  :type Stairs)
   (:name ?door1    :type Door)
   (:name ?sign2    :type Sign)
   (:name ?railing3 :type Railing)
   (:name ?canopy4  :type Canopy))
  :restrictions
  ((:name ?bn-s-d   :relation belowNeighbor
    :subject ?stairs0 :object ?door1)
   (:name ?b-s-c    :relation below
    :subject ?stairs0 :object ?canopy4)
   (:name ?o-s-r    :relation overlap
    :subject ?stairs0 :object ?railing3)
   (:name ?bn-s-s   :relation belowNeighbor
    :subject ?stairs0 :object ?sign2)
   (:name ?an-d-s   :relation aboveNeighbor
    :subject ?door1 :object ?stairs0)
   (:name ?bn-d-c   :relation belowNeighbor
    :subject ?door1 :object ?canopy4)
  :discriminators
   ((?bn-s-d) (?b-s-c) (?o-s-r)
    (?bn-s-s) (?an-d-s) (?bn-d-c))))
```

Figure 2. Local aggregate description. A general `Scene-Aggregate`, which specifies restrictions that hold for all aggregates and a specific `Entrance` aggregate that inherits the general restrictions and properties. Question marked symbols (e.g. `?an-d-s`) indicate variables, which can bind objects or relations.

In Figure 3 the physical parameters and relations are listed, which are used in the following domain of Scene Interpretation.

4 Analysis of Aggregation Processing and Representation with a Configuration Language

In this section, we identify all key situations that may occur during the processing of restricted aggregates. This is done by permuting possible variabilities given by restricted aggregate models. Furthermore, for each key situation we provide representations based on facilities given by a configuration language.

As described above, depending on the given information about aggregates and parts, certain activities should be performed by the configuration process. For an aggregation, a key situation can be characterized by the presence or absence of an aggregate A instance (e.g. a

[5] Discriminators should not be confused with mandatory or optional parts of an aggregate. In the example, all parts (*door*, *stairs*, and *canopy*) are mandatory for an *entrance*. For the existence of an entrance not all of them have to be observed, but only those mentioned in the discriminators. However, all mandatory parts will be created as hypotheses once the aggregate is instantiated.

Spatial relations reflect the appropriate geometric relation: `overlap`, `inside`, `left-of`, `right-of`, `above`, `below`, `top-left`, `right-left`, `bottom-left`, `right-left`, `left-neighbor`, `right-neighbor`, `above-neighbor`, `below-neighbor`, `check-variability`.
Geometric parameters for describing bounding box and size of an object: `pos-x-1`, `pos-y-1`, `pos-x-2`, `pos-y-2`, `size-x`, `size-y`.
Spatial predicates check the geometric parameters, whether the spatial relations hold: `overlap-p`, `inside-p`, etc.

Figure 3. Predefined spatial relations, geometric parameters, and spatial predicates on geometric parameters for the Scene Interpretation domain. The `x-neighbor` relations indicate direct neighbors in the mentioned x direction.

No.	compositional rel. for A and p_i	spatial rel. for p_i	geometric para. for p_i
1	original	given	original
2	original	original	given
3	given	original	original
4	given	given	original
5	given	original	given
6	original	given	given
7	given	given	given

Figure 4. Possible situations for instances of one aggregate A and for potential parts p_i. If not given as constant or reduced, the value is an unchanged interval or not yet computed relations of the concept (indicated by $original$).

`balcony`) and its parts p_i (e.g. a `door`, a `window`). Hence, the variability of a key situation can be described as follows:

Instances A and p_i might be or might not be given. If neither A nor p_i are given, A may be constructed from the always given goal object g. This means, that an instance of a certain concept is created where all properties are original. If only A is given, appropriate p_i have to be constructed analogously and vice versa.

Concept type of A may be of a taxonomical *leaf* concept type or a *specializable* concept type. A *leaf* concept cannot be specialized further, and thus, only the aggregate parts and restrictions have to be computed. If a *specializable* concept type is given (e.g. as a general type like `Facade-Object`), a more specific type has to be computed by considering given or possible parts of the aggregate including their restrictions. For example, an instance of `Facade-Object` may be specialized to `Balcony`, if it has an instance of `Door` as a part.

Concept type of p_i is analogous to the above considerations.

Compositional relation between A **and** p_i might be or might not be given. If such relations are given, the restrictions given by A have to be checked for p_i. If the compositional relations are not given, they have to be established, if the p_i fulfill the restrictions given in the model of A. If A has further potential parts than p_i, those have to be created (i.e. part *hypotheses* are created, see [10]) and have to fit the given p_i.

Spatial relations between p_i might be or might not be given. If they are given, the geometric parameters have to have values according to the spatial relations, an appropriate aggregate has to be created (i.e. an aggregate *hypothesis*), and the compositional relations of that aggregate to the p_i have to be established. If spatial relations are not given, they must be computed from the spatial predicates and geometric parameters.

Geometric parameters of p_i and A might be or might not be given. If given, the appropriate spatial relations can be computed. If they are not given, geometric parameters can be computed from spatial relations.

Expecting that the instances with appropriate concept types are given, Figure 4 shows all other situations. Those are discussed in the following.

```
(define-conceptual-constraint
  :name Spatial-relation-from-predicate
  :structural-situation
  ((:name ?o1 :type Scene-Object)
   (:name ?o2 :type Scene-Object
              :relations
                ((self
                   #'(aboveNeighbor-p *it* ?o1)))))
  :action-part
  ((ensure-relation
      (above-neighbor ?o2 ?o1)
      (below-neighbor ?o1 ?o2))))
```

Figure 5. Generic conceptual constraint for Case 2 and Case 5 (*Case-Gen-1*). By checking spatial predicates the explicit spatial relations are established. `*it*` refers to the value of the relation, in the case of `self` to the object bound to `?o2`.

Each case has distinct impacts on the configuration process, i.e. distinct activities have to be performed. In general, in each case the missing information (in Figure 4 indicated by $original$) has to be computed from the fixed or reduced ones (indicated by $given$). Besides these activities, also the mappings to the representation facilities of the configuration language have to be specified. In the following, for each key situation representations are given as reasoning chunks.

```
(define-conceptual-constraint
  :name Spatial-relation-from-relation
  :structural-situation
  ((:name ?o1 :type Scene-Object)
   (:name ?o2 :type Scene-Object
              :relations
                ((above-neighbor *it* ?o1))))
  :action-part
  ((less (y-pos-2 ?o1) (y-pos-1 ?o2))
   (less (x-pos-1 ?o2) (x-pos-1 ?o1))
   (greater (x-pos-2 ?o2) (x-pos-2 ?o1))))
```

Figure 6. Generic conceptual constraint for Case 1 and Case 4 *Case-Gen-2*. By checking spatial relations the geometric parameters are computed by numeric constraints.

Case Reduction: From Case 2 to Case 6, and from Case 5 to Case 7. The mapping of given geometric parameters of p_i to spatial relations can be done with one generic conceptual constraint (*CC-Gen-1*) that holds for all types of parts (see Figure 5). There, arbitrary `Scene-Objects` (the super concept of every part or aggregate) are checked with spatial predicates in the structural situation and the appropriate spatial relations are established by `ensure-relation` in the action part. Because the spatial predicates can handle fixed and reduced values, this mapping is straight forward. With this conceptual constraint, also Case 2 and Case 5 can be processed as Case 6 and Case 7, respectively.

Case Reduction: From Case 1 to Case 6, and from Case 4 to Case 7. Similarly Case 1 and Case 4 can be reduced by introducing one generic conceptual constraint for mapping spatial relations to geometric parameters (see Figure 6, *CC-Gen-2*). The condition matches all p_i that are in the modeled spatial relation. The action part uses numeric constraints to compute the geometric parameters. However, in this case, because the geometric parameters are original, the mapping only reduces their intervals according to the spatial relations. This is done internally by constraint actions because of the underlying mathematics (see Section 2).

Case Reduction: From Case 6 to Case 7. The *compositional relation* of the parts to an aggregate instance have to be established. Each aggregate concept has to be checked that might be able to have parts with the given spatial relations. This can be achieved by using a conceptual constraint as shown in Figure 7. The structural situation of such a conceptual constraint describes the spatial relation which holds between the parts. The action part uses the `create-instance` constraint operation, which selects an aggregate concept out of a set of concepts. This set represents all aggregates that may have parts with the given spatial relations. According to the given concepts, one aggregate type A_s is selected (i.e. by considering probabilities). A new instance of A_s is created and the compositional relations between this new instance and the p_i are established. For every discriminator of an aggregate (in Figure 2 every spatial relation), one conceptual constraint of this kind is modeled (*Case-6-ccs*). Thus, this conceptual constraint can handle all situations where some parts are given which are not yet part of an aggregate.

```
(define-conceptual-constraint
  :name Entrance-creation
  :structural-situation
  ((:name ?stairs0 :type Stairs
          :relations
            ((part-of #'(free-p *it*))))
   (:name ?door1 :type Door
          :relations
            ((part-of #'(free-p *it*))
             (above-neighbor ?stairs0))))
  :action-part
  ((create-instance (Entrance Terrace)
                    (part-of ?stairs0)
                    (part-of ?door1))))
```

Figure 7. Conceptual constraints for creating a compositional relation with a new aggregate (i.e. an example for a *Case-6-ccs*). The conceptual constraint matches all stairs and doors that are appropriately related and are not yet part of an aggregate (indicated by $free - p$). `create-instance` selects one appropriate aggregate (e.g. the most probable one), creates one instance of that aggregate (e.g. `Entrance`), and establishes the compositional relations.

Case 3: Only compositional relations between A and p_i are given. The spatial relations between p_i have to be computed from the given compositional relations. Here the conceptual constraints (*Case-3-ccs*) have the following form: The structural situation checks the compositional relation between the A and p_i. The action part establishes the spatial relations between the p_i. For each spatial relation of an aggregate one conceptual constraint of the form illustrated in Figure 8 is created.

Case 7: Compositional relations between A and p_i, spatial relations p_j and geometric parameters of p_j are given. All conceptual constraints match and check the given relations. If the compositional relations are fixed for the same parts as the spatial relations (i.e. if $p_i = p_j$) the *Case-3-ccs* can be used for ensuring the spatial relations - `ensure-relation` than only checks the spatial relations between the parts. *CC-Gen-1* computes the geometric parameters for the p_j.
If there exists a p_j which is not yet part of A (i.e. its `free-p`), but holds the spatial relations of A, the compositional relation can be established with a further type of conceptual constraint (*Case-4-ccs*, see Figure 9). Furthermore, there may be combinations where one part is already part of A and another is not, e.g. in Figure 9 `Stairs` may be part of Entrance while `Door` is not, or vice versa. For this reason, the predicate `free-or-in-agg-p` is introduced that checks, whether an object is part of no aggregate or of the indicated one (e.g. part of `Entrance`).

Thus, five different types of reasoning chunks are finally identified:

```
(define-conceptual-constraint
  :name Entrance-Spatial-relation
  :structural-situation
  ((:name ?e :type Entrance)
   (:name ?stairs0 :type Stairs
          :relations ((part-of ?e)))
   (:name ?door1 :type Door
          :relations ((part-of ?e))))
  :action-part
  ((ensure-relation
      (above-neighbor ?door1 ?stairs0)
      (below-neighbor ?stairs0 ?door1))))
```

Figure 8. Conceptual constraint for Case 3 (*Case-3-ccs*).

```
(define-conceptual-constraint
  :name Entrance-Spatial-relation
  :structural-situation
  ((:name ?stairs0 :type Stairs)
   (:name ?door1 :type Door
          :relations
            ((aboveNeighbor ?stairs0)))
   (:name ?e :type Entrance
          :relations
            ((has-parts
               #'(free-or-in-agg-p ?stairs ?doors1 *it*)
               #'(check-variability *it*
                          ?stairs0 ?door1)
               #'(check-bounding-box *it*
                          ?stairs0 ?door1)))))
  :action-part
  ((ensure-relation
      (part-of ?stairs0 ?e)
      (has-parts ?e ?stairs0))
   (ensure-relation
      (part-of ?door1 ?e)
      (has-parts ?e ?door1))))
```

Figure 9. Conceptual constraint for Case 4 (*Case-4-ccs*). The predicate `free-or-in-agg-p` checks, whether the objects are parts of no aggregate or already part of the aggregate. `check-variability` and `check-bounding-box` are further aggregate restrictions that have to hold.

1. Mapping between numeric parameters and explicit relations (quantitative/qualitative mapping). This mapping is done by the conceptual constraint *Case-Gen-1*.
2. Mapping between explicit relations and numeric parameters (qualitative/quantitative mapping). This mapping is done by the conceptual constraint *Case-Gen-2*.
3. Creating new aggregates from given discriminative explicit relations (discriminators). This mapping is done by one conceptual constraint for each discriminator of an aggregate (*Case-6-ccs*).
4. Checking if restrictions between parts hold for parts that are elements of an aggregate. This mapping is done by one conceptual constraint for each discriminator of an aggregate (*Case-3-ccs*).
5. Integrating appropriate parts in existing aggregates while considering aggregate restrictions. This mapping is done by one conceptual constraint for each discriminator of an aggregate (*Case-4-ccs*).

5 Experiments in the Scene Interpretation Domain

We tested the previously described representation for the construction of descriptions of facade scenes. In Figure 10, left, primitive parts like *windows*, *stairs* are shown. These parts are aggregated to *balconies* and an *entrance* (see Figure 10, right). The experiments further showed that the selection of the domain-dependent predicates like `check-variability` and appropriate discriminator (see Figure 2) are very important. If several aggregates of the same type have to be considered (see Figure 11, right), the identification of the corresponding primitive parts are computed by those predicates.

6 Discussion and Summary

The generic aggregation reasoning chunks presented in this paper have the following properties:

- They distinguish between numeric, quantitative parameters typically given in databases or sensors, and qualitative relations which are used in abstract aggregation models.
- They compute all kinds of entities, i.e. aggregates, parts, relations and parameters depending on the given information.
- They are generic, i.e. they do not depend on the domain used in the examples, but can be applied to any domain with restricted aggregates, e.g. also to domains with temporal relations.

Because of the expressive language used in configuration technologies, domain restrictions with numeric, interval-based constraints and n-ary constraints can be used. Also, the configuration language with concepts and constraints used in our work can be mapped to similar representation facilities like classes, rules, and functions of other configuration approaches. Thus, the paper provides general modeling and representation facilities used for composing restricted aggregates.

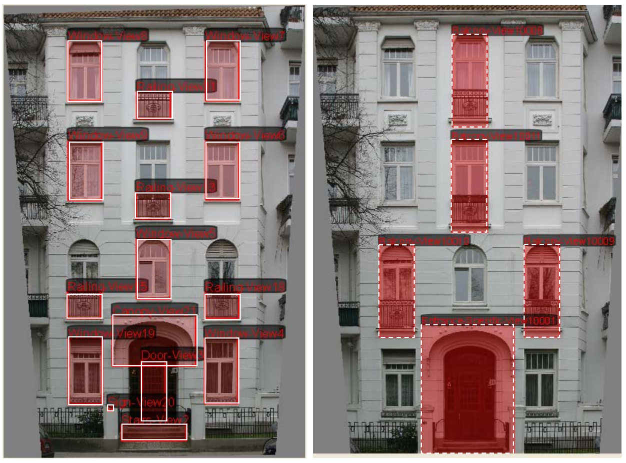


Figure 10. Left: Primitive facade objects here provided by annotation. Right: Constructed aggregates of type `entrance` and `balcony`. The annotated primitives and the automatically created aggregates are highlighted for presentation reasons.

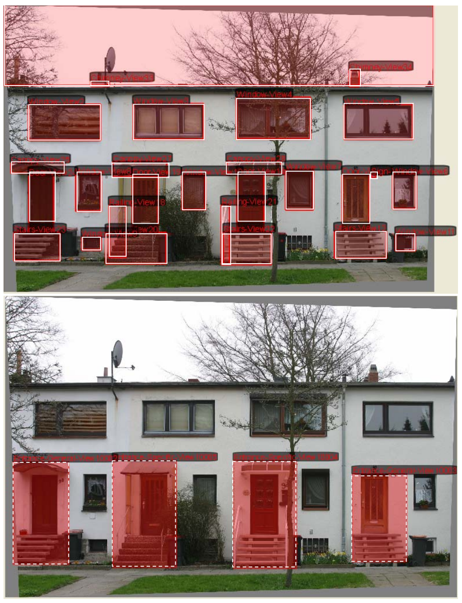


Figure 11. Further example with primitives and several aggregates of one type.

ACKNOWLEDGEMENTS

This research has been supported by the European Community under the grant IST 027113, eTRIMS - eTraining for Interpreting Images of Man-Made Scenes.

REFERENCES

[1] R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode, 'PLAKON - an Approach to Domain-independent Construction', in *Proc. of Second Int. Conf. on Industrial and Engineering Applications of AI and Expert Systems IEA/AIE-89*, pp. 866–874, (June 6-9 1989).
[2] A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.
[3] A. Günter and L. Hotz, 'KONWERK - A Domain Independent Configuration Tool', *Configuration Papers from the AAAI Workshop*, 10–19, (July 19 1999).
[4] A. Günter and C. Kühn, 'Knowledge-based Configuration - Survey and Future Directions', in *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, ed., F. Puppe, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg, (March 3-5 1999).
[5] A. Haag, 'Sales Configuration in Business Processes', *IEEE Intelligent Systems*, (July/August 1998).
[6] J. Hartz and B. Neumann, 'Learning a knowledge base of ontological concepts for high-level scene interpretation', in *International Conference on Machine Learning and Applications*, Cincinnati (Ohio, USA), (December 2007).
[7] M. Heinrich and E. Jüngst, 'A Resource-based Paradigm for the Configuring of Technical Systems from Modular Components', in *Proc. of 7th IEEE Conf. on Artificial Intelligence for Applications (CAIA'91)*, pp. 257–264, Miami Beach, Florida, USA, (February 24-28 1991).
[8] L. Hotz, 'Configuring from Observed Parts', in *Configuration Workshop, 2006*, eds., C. Sinz and A. Haag, Workshop Proceedings ECAI, Riva del Garda, (2006).
[9] L. Hotz and B. Neumann, 'SCENIC Interpretation as a Configuration Task', Technical Report B-262-05, Fachbereich Informatik, University of Hamburg, (March 2005).
[10] L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuis, and J. MacGregor, *Configuration in Industrial Product Families - The ConIPF Methodology*, IOS Press, Berlin, 2006.
[11] M. Jing and H. Boley, 'Interpreting SWRL Rules in RDF Graphs', *Electronic Notes in Theoretical Computer Science*, **151**, 53–69, (2006).
[12] D. Margo and P. Torasso, 'Interactive Configuration Capability in a Sale Support System', in *Proc. of Configuration Workshop, 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pp. 57–63, Seattle, USA, (August 2001).
[13] J. McDermott, 'R1: A Rule-based Configurer of Computer Systems', *Artificial Intelligence Journal*, **19**, 39–88, (1982).
[14] S. Mittal and F. Frayman, 'Towards a Generic Model of Configuration Tasks', in *Proc. of Eleventh Int. Joint Conf. on AI IJCAI-89*, pp. 1395–1401, Detroit, Michigan, USA, (1989).
[15] S. Pribbenow, 'What's a Part? - On Formalizing Part-Whole Relations', in *Foundations of Computer Science*, volume 1337 of *Springer Lecture Notes in Computer Science*, pp. 399–406, (1997).
[16] K.C. Ranze, T. Scholz, T. Wagner, A. Günter, O. Herzog, O. Hollmann, C. Schlieder, and V. Arlt, 'A Structure-based Configuration Tool: Drive Solution Designer DSD', *14. Conf. Innovative Applications of AI*, (2002).
[17] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, 'Towards a General Ontology of Configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), 12*, 357–372, (1998).
[18] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, **10(2)**, 111–126, (1997).
[19] A. Varzi, 'Parts, Wholes, and Part-Whole Relations: The Prospects of Mereotopology', *Data and Knowledge Engineering*, **20(3)**, 259–286, (1996).
[20] D.L. Waltz, 'Understanding Scenes with Shadwos', in *The psychology of computer vision*, pp. 19–91, New York, (1975). McGraw-Hill.
[21] P.H. Winston, R. Chaffin, and D. Herrmann, 'A Taxonomy of Part-Whole Relations', *Cognitive Science*, **11**, 417–444, (1987).
[22] B. Yu and H.J. Skovgaard, 'A Configuration Tool to Increase Product Competitiveness', *IEEE Intelligent Systems*, **13**(4), 34–41, (July 1998).

A Generative Constraint Model for Optimizing Software Deployment[1]

Mihai Nica and **Bernhard Peischl** and **Franz Wotawa**[2]

Abstract. In this article we propose a constraint model for automated software deployment for embedded automotive systems. Unlike to specific algorithmic approaches, our purely model-based approach is applicable in an early system development stage as there is no need to obtain specific measurements from a prototype. Besides of this advantage, the constraint model offers unprecedented flexibility: By taking account of generative constraints, we not solely extend the application scope of CSPs but provide novel model enhancements that allow for further optimizations. Notably, the presented model extension allows for capturing synchronous and asynchronous ECU tasks and is detailed enough to capture the (optimal) alignment of function blocks to ECU tasks under presence of rigorous point to point timing constraints. The article thus further motivates the application of CSPs in engineering embedded automotive software.

1 Introduction

Today's upper class cars contain up to 80 ECUs (Electronic Control Units), several bus systems, and about 55 percent of all failures are caused by electronics, software, cables and connectors [1], [2]. More and more functions in today's cars involve electronics and software, 80-90 percent of the new innovative features are realized by distributed embedded systems. Following this mainstream trend, even highly safety critical mechanical and hydraulic control systems will be replaced by electronic components.

In recent years, the focus in engineering embedded automotive systems has been on rather detailed abstractions primarily dealing with implementation related issues like models for code generation. Model-based optimization techniques typically take a back seat in the overall design process since they lack suitable, standardized notations, methodologies, and integration into the model-driven tool chain.

As today's embedded automotive software is highly distributed, the automotive industry devotes increasing efforts to develop tools for automated software deployment [3]. The underlying foundations comprise techniques like genetic algorithms and various other clustering techniques [3]. However, to our best knowledge, none of the current approaches addresses automated software deployment in terms of a model-based approach. Relying on an algorithmic approach one has to perform measurements to obtain meaningful metrics for certain parameters as, for example, a reference value for the bus load. Besides of the provision of a prototype for measurements, this considerably hampers the seamless integration into the model-based development paradigm.

In [17] we address the prevalent complexity of automated software deployment in a resource constrained setting even catering stakeholders at an early development stage. In this article we extend the CSP model presented in [17] by incorporating (1) rigorous point-to-point timing requirements. Moreover, we propose a model enhancement for (2) capturing the optimal alignment of function blocks to synchronous or asynchronous tasks. Notably, as generative constraints typically provide even more optimized solutions with reasonable computational effort, (3) we illustrate how to employ this powerful technique to further optimize our software deployment approach.

This article is organized as follows. Section 2 briefly introduces CSPs and, in particular, generative constraints. Afterwards, in Section 3, we briefly introduce the problem of automated software deployment on a conceptual level and present a general CSP model capable of handling resource -, quality-, cost-, and timing constraints in Section 4. In this section we also outline how to deal with point-to-point timing constraints and address the alignment of function blocks to specific (asynchronous as well as synchronous) tasks. Notably we exploit the powerful mechanism of generative constraints to achieve an optimal solution. Section 5 discusses related work and finally concludes this article.

2 Constraint Satisfaction Problems

Constraints systems are a natural and straightforward way of describing specifications and requirements for hardware and software systems. A *Constraint Satisfaction Problem* (CSP), (V, D, CO), is characterized by a set of variables $V = (v_1, ..., v_n)$, each variable having a domain D, and a set of constraints $CO = (c_1, ..c_k)$ which defines a relation R between variables. The variables in a relation $R \in CO$ are called the scope S_R of the relation. There are very effective reasoning algorithms available for CSP, e.g., for computing solutions. A solution of a CSP is an assignment of values to the CSP's variables which does not contradict any given constraint. State of the art constraints solver are available for solving CSPs. More information about CSPs can be found in Rina Dechter's book on constraints [5].

Due to complex specifications, it is possible for the CSP associated to a given problem to find itself in an *inconsistent state*. We say that a CSP, (V, D, CO), is in an inconsistent state, if for its corresponding variables set $V = (v_1, ..., v_n)$, there exists no valid initialization set from the domain D such that all the constraints from the constraints set $CO = (c_1, ..c_k)$ are simultaneously fulfilled. A CSP is *consistent*

[1] The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsfrderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT) and by the Austrian Science Fund (FWF) under grant P20199-N15

[2] Authors are listed in alphabetical order, Austria, email: {nica,peischl, wotawa}@ist.tugraz.at

when there exists at least one valid initialization of the variables set V such that all constraints from the set CO are simultaneously fulfilled.

Inconsistencies can be overcome ***by relaxing the constraints system***. Within our algorithm there are two possible situations when the CSP can find itself in an inconsistent state:

1. *Resource Constraints Inconsistency.*This type of inconsistency can originate from two situations. The first situation is when attempting to deploy a cluster on an ECU which dose not provide enough resources for a safe execution of the function blocks. By means of a generative approach, we identify thous resources that lead to the inconsistent state and safely generate a new type of ECU that overcomes thous problems. The second situation is when, after computing all possible combination of deploying the function blocks, we cannot satisfy the required level of bus load. This situation can not be directly overcome by our algorithm, and leaves the decision of *what is better to do*, to the developer.
2. *Price Constraints Inconsistency.*This inconsistency appears when the desired all-around cost of the system can not be accomplished when trying to fulfill all of the quality requirements. Again the decision of eliminating the inconsistency lies within the responsibility of the developers.

Sometimes we want to extend the system's functionalities or insert new components into it. In order to do that we need to expand the current consistent CSP configuration. We do this by means of generative constraints. However,the CSP can now find itself in an inconsistent state. This is why after each modification undertaken to a consistent CSP, a consistency check has to be performed.

The consistency check mechanism verifies if a given CSP is in an inconsistent state. There are several consistency check algorithms that can be successfully applied. The most popular of them are arc-consistency, path consistency and n-consistency check. A description of these algorithms is found in [5]. An optimized algorithm for consistency check is the Max-Restricted Path Consistency algorithm [9].

2.1 Generative Constraints

The definition of a generative constraints system is given in [16]. The paper states that a generative constraint satisfaction problem is a tuple $GCSP(X_0, \Gamma, T, \Delta_0)$ where X_0 is the initial variable set, Γ the set of generative constraints, T the set of variables types and Δ_0 is a relation that has the property that each of her tuples $(x, (t, i))$ associate a variable x from the set of variables to its type t and to a unique identifier that indicates that x is the i-th variable of type t [16] states that a *generic constraint* is a constraint over meta variables, where the meta variables are replaced, after the preprocessing step is over, with concrete variables having the same type as the meta variables. A *meta variable* M^i of type t is a place-holder for all concrete variables of type t.

3 Problem Statement

By *partitioning* we understand the process of dividing a set of function blocks into groups of subsets of function blocks. Each group of function blocks is executed on an available control unit (CU). We denote such a group as *cluster*. By function block we understand a specific task that has to be executed by the system such that a certain required functionality can be provided. For example the signaling function of a car can be abstractly seen as a 3-tasks set: switch on the signal-commutator, signaling an interrupt to the signal-light controller and switching on the signaling light.

The function blocks deployed on a control unit exchange informations with the function blocks from other control units by means of the main data bus. How the function blocks are partitioned into clusters is decisive for the performance of the entire system. A data-bus load of more than 70% is already critical and its an indicator for a bad partioning schema. Several algorithms were proposed for computing the partitioning schema of a set of function blocks [4].

We propose a partitioning algorithm based on the CSP representation of the system requirements and on the quality specifications. Moreover we use generative constraints in order to expand our initial CSP such that it can incorporate each new added component.

In order to build the CSP we need to define the system's parameters. We configure the software deployment strategy for an automotive system. Within the automotive industry an electronic control unit (ECU) is the computational part on which the software functions associated to different requirements of the system, e.g. DVD-play, ABS, torque vectoring, or control of the attitude angle, are executed. The problem of partitioning into clusters is equivalent to the problem of assigning each ECU a set of function blocks that are periodically executed on it.

In order to exchange data, the function blocks have to communicate with each other. We can see the set of the function blocks as a network where the nodes are the functions. Each connection that is established between two function blocks has associated an weight. This denotes the communication frequency between the connected functions. One criteria of the partitioning algorithm is taking into account this weights. It is recommended to group together thous function blocks that communicate very often.

4 A Constraint Model for Software Deployment

The system's final CSP represents a union of three categories of constraints:

1. *Resource Constraints*: The resources of the CU, on which the cluster is executed, give us the resource constraints system. The memory of the CU and the processing power, are criteria which impose restrictions on the clusters that can be executed on the given CU.
2. *Quality Constraints*: Using quality functions we define the quality constraints. They assure that the system will behave within the given quality criteria. For example a quality criteria is a bus load that is always under 40%.
3. *Cost Constraints*: The cost constraints are given by the implementation's cost of the CUs. There can be more types of CUs with different properties and different implementation costs. It is possible that although a certain CU is expensive to implement it offers an all around smaller cost than when using 10 CUs that perform the same task. An optimal cost is hard to achieve. These type of constraints are strongly connected with an arbitrary parameter that we call *desired general cost* (DGC). We define the cost constraints such that they always assure that the 'all around system's costs' is smaller than the DGC. We also try to have the costs as low as possible without cutting off the system's performance.
4. *Time Constraints*: Within an automotive system each software functionality has to be executed within an amount of time. It will be a catastrophe if the braking function would take 30s to execute. When partitioning function blocks into clusters this criteria must also be considered with respect to the ECU on which the deployment is made.

We use the following set of definitions:

Definition 1 (Function Block) *Any function block (of t function blocks) is associated with a unique identifier f_i, its processing requirements $pow(f_i)$, the memory requirements $mem(f_i)$, and the worst case execution time $w_{tc}(f_i)$.*

Definition 2 (ECU) *Every Electronic Control Unit ECU_i is associated with a processing capacity max_{ECUpow_i}.*

Definition 3 (Bus System) *Every bus system B is associated with a worst case execution time. We assume a function $t_{wc}(B)$ returning the worst case execution time given a bus system B.*

Definition 4 (Point-to-Point Requirement) *Any temporal requirement $Req_{(i,j)}$ specifies the maximal execution time that is allowed between the source i and the sink j.*

Definition 5 (Gateway) *A gateway G connects two different bus systems. For the underlying conversion process we assume a worst case execution time. Given an gateway G, a function $t_{wc}(G)$ returns this worst case execution time.*

4.1 Resource Constraints

We start building the *resource constraints system.*

1. The overall memory consumption of the function blocks is smaller or equal to the available memory. Usually not all function blocks are executed in the same time, but in the worst case scenario, this trivial safety constraint assures us that no jamming occurs in the function execution process.
$\sum_i mem(f_i) \leq \sum_j max_{ECUmem_j}$
2. An adjacent memory constraint is the *maximal function block memory constraint.* That is, let f_{max} be a function block such that the memory requirement of f_{max}, $mem_{f_{max}}$, is the maximum from all function's memory requirements. There exist an ECU, $ECU_k \in ECU$, with the available memory mem_k, such that $mem_k \geq mem_{f_{max}}$.
3. After we decide to deploy a cluster of functions, $C_j = \{f_i...f_{i+n}\}$,$i \geq 1$, on an ECU, ECU_j, then ECU_j must provide enough memory and processing power to host the deployed functional blocks. The function $deploy(ECU_j)$ returns the indices of the function blocks deployed on ECU_j.
$\sum_{i \in deploy(ECU_j)}(mem(f_i) \leq max_{ECUmem_j}) \wedge$
$\sum_{i \in deploy(ECU_j)}(pow(f_i) \leq max_{ECUpow_j})$
4. A function block is deployed on a single ECU only.
$\forall i, j \in \{1..n\}, i \neq j \cdot deploy(ECU_j) \cap deploy(ECU_i) = \emptyset$
5. Any function $deploy$ that distributes all functional blocks f_i on max ECUs is a solution.
$\{1..n\} = \bigcup_{j=1}^{max} deploy(ECU_j)$

By unifying the above constraints system we derive the resource constraints system (RCS):

$$RCS : \left\{ \begin{array}{c} 1. \sum_{\mathtt{i}} \mathtt{mem(f_i)} \leq \sum_{\mathtt{j}} \mathtt{max_{ECUmem_j}}; \\ 2. \exists f_i | f_i \in F, i \in [1,t], \forall j \in [1,t], \\ i \neq j, mem_{f_i} \geq mem_{f_j} \\ \Rightarrow \exists ECU_l \in ECU, l \in [1,k] : \\ mem(ECU_l) \geq mem_{f_i}; \\ 3. \sum_{i \in deploy(ECU_j)}(mem(f_i) \leq max_{ECUmem_j}) \wedge \\ \sum_{i \in deploy(ECU_j)}(pow(f_i) \leq max_{ECUpow_j}); \\ 4. \forall i, j \in \{1..n\}, i \neq j, deploy(ECU_j) \\ \cap deploy(ECU_i) = \emptyset; \\ 5. \{1..n\} = \bigcup_{j=1}^{max} deploy(ECU_j); \end{array} \right\}$$

4.2 Timing Constraints

Regarding the timing requirements, for sake of simplicity, we assume that there is only a single path from the source x to the sink y and that there are no loops in the network. Under these assumptions the point-to-point timing requirements map to a constraint model as follows.

1. $t_{xy} \leq \sum_i t_{wc}(f_i) + t_{wc}(B_i) + t_{wc}(G_i)$.

For every timing requirement $Req_{(a,b)}$ we instantiate these constraints.

A system's software functionality is divided in a finite number of tasks (software blocks) that have to be sequentially executed. Each ECU executes in one execution cycle a number of tasks. This tasks are not necessarily part of the same software functionality and have to be periodically executed. Each software functionality has to be executed within a certain prior known time. Because of that, time requirements with regard to the task execution are enforced. From this we extract the *time constraints system* (TCS) of the system.

Given a software functionality SF consisting of a sequential set of tasks $T = \{t_1, ..., t_k\}$ and a required maximum execution time T_{max}. Then if function $time(t_i)$ with $i \in \{1, .., k\}$, returns the time necessary to execute task t_i on the assigned ECU then $\sum\limits_i time(t_i) \leq T_{max}$. This constraint is verified after all tasks t_i of the software functionality SF are deployed.

Within an ECU, each task as an assigned execution time within the execution cycle. It possible that a task is completely executed after more execution cycles. For example, if for a task t_i we need 4 ms, but the execution slice for this task is of 1s, we need 4 execution cycles in order to complete the task. If one cycle takes 10ms to complete, then, if the task is programed to be executed first on the execution cycle, we need 31s in order to execute it. This can lead to a violation of the global time assigned for the software functionality which includes this task. We have to be careful not to neglect this possible inconsistency. Let t_{ECUi} be the time that a ECU needs in order to execute a task t_i, then $t_{ECUi} \leq time(t_i)$.

An *asynchronous task* of a software functionality is not programed to be periodically executed and is triggered by special events from the outside world. Each ECU has in its execution cycle a slice that is specially assign to this type of tasks. The slice has a fixed amount of time in which it can execute the asserted task. The possible asynchronous tasks are known for each type of software functionalities, e.g for braking we have as possible asynchronous task the ABS functionality; triggered only in special cases. The asynchronous tasks are also included in the partitioning process. The constraints associated to this type of tasks are the same as in the case of synchronous tasks.

4.3 Quality Constraints

The *quality constraints system* are the most important factor when we partition the function blocks into clusters. In order to build these constraints system we use a set of functions, named *quality functions*. The quality functions offer us a metric for computing the optimal partitioning of the function blocks. The constraints are created by imposing output values that these functions should not exceed for a given cluster. The constraints solver tries to find a set of function blocks such that all the quality constraints are fulfilled. When it finds such a set it creates the cluster.

Besides, as an extra quality constraint, we try to keep the output values of the quality functions to a level close to optimal(such that the cost are minimal). Each quality function receives as input parameter the CF set. How this set is build depends on the user and on the described system. There are more solutions proposed for building this set; one, given in [4], proposes a representation of the CF set by means of a geometrical matrix. It is beyond the scope of this paper to discuss how the CF is created. We presume that the set is already given and use it directly as input for the quality functions.

We build the quality constraints system based on the quality functions set. We use the quality functions presented in [4].

We define the following:

Definition 6 (Cluster's external cost) *It represents the frequency with which the function blocks within a cluster C_i,$i \in [1,c]$,communicate with the rest of the function blocks from the network. We denote this metric trough E_i and we compute it as the average CF between the function blocks within the cluster and the external function blocks.*

Definition 7 (Cluster's internal costs) *It represents the frequency with which the function blocks communicate with each other within a given cluster C_i,$i \in [1,c]$. We denote this metric by I_i and it represents the average of all CF within the cluster C_i.*

Definition 8 (Cluster's diameter) *It represents, based on the CF of the function blocks, the average distance between the function within a given cluster C_i,$i \in [1,c]$. We denote this metric through $diamC_i$.*

Definition 9 (Distance between Clusters) *It represents, based on the CF of the function blocks, the average distance between a cluster C_i and a cluster C_j,$i,j \in [1,c]$, $i \neq j$. We denote this metric by $d(C_i, C_j)$.*

Definition 10 (External costs between clusters) *It represents, based on the CF of the function blocks, the external cost between a cluster C_i and the function blocks of a cluster C_j,$i,j \in [1,c]$, $i \neq j$. We denote this metric by $E(C_i, C_j)$.*

Definition 11 (Cluster's Nodes) *It represents the number of function blocks within a cluster C_i,$i \in [1,c]$. We denote this metric by N_i.*

The quality functions are defined below. Detailed informations about these functions can be found in [4].

1. *The External-Internal Ratio* is a ratio between the external and the internal costs must be as low as possible. That is, a good cluster is a cluster which communicates as little as possible with the other function blocks from the network and that has the internal communication frequency as high as possible. We define for every cluster a communication ratio limit, CRL_{max}, which represents the qualitative limit that every cluster must respect.
$\forall C_i, i \in [1,c]\ \frac{E_i}{I_i} \leq CRL_{max}$
2. *The Davies Bouldin Criteria* shows a good partitioning when the factor is as low as possible. The Davies Bouldin (DB) factor is computed only after all the cluster are formed. We set a limit, DB_{max} that should never be surpass by the final cluster partitioning. After computing all the clusters c, we compute the DB factor. If it is greater than DB_{max} then the constraint is violated and a new partitioning of the function blocks is performed. If the constraint holds a valid configuration with respect to the DB factor was found.
$DB = \frac{1}{c}\sum_{i=1}^{c} max_{j\neq i}\left[\frac{diam(C_i)+diam(C_j)}{d(C_i,C_j)}\right]$
3. *The Modularization Factor* (MF) is an indicator of a compact clustering of the function's blocks. The value of this factor should be as high as possible. For our constraints system we settle a minimal value, MF_{min},below which the optimality criteria is violated. If, after computing the all clusters, we observe that the value of MF is smaller than MF_{min}, then the constraint is violated and we discard the partitioning. If the value of MF is greater than MF_{min} then we found a valid solution.
$$MF = \frac{\sum_i I_i}{\sum_i \frac{N_i(N_i-1)}{2}} - \frac{\sum_{i<j} E(C_i,C_j)}{\sum_{i<j} N_i N_j}$$
4. *The SILHOUETTE factor* (Sh) verifies the correctness of the distribution of a function f_i within a cluster C_i with respect to a neighbor node C_j. The domain of the Sh value of the function f_i is $[-1,1]$. A good distribution of the functions f_i within a cluster C_i, has the Sh value in the vicinity of 1. For every function f_i, we compute $Sh(f_i)$. If this value diverges with more than δ_{max} from 1 then the constraint is violated, the function is not distributed within cluster C_i and we start the search for a new cluster.
$Sh(f_i) = \frac{d(f_i,C_j)-d(f_i,C_i)}{max(d(f_i,C_j),d(f_i,C_i))}$
5. *The Cluster Load Deviation* (CLD) is computed after all the clusters c are created. Small values of this function denote a good partitioning of the function blocks. In a good case scenario all the clusters have a similar number of function blocks within them. We have the following constraint: the final CLD value of the network must not be greater than an optimal criteria CLD_{max}. If the CLD of the network is greater than CLD_{max} the partitioning of the function blocks is discarded and we restart the partitioning process. If the value of CLD is smaller than CLD_{max} then we have found a valid partitioning.
$$CLD = \sqrt{\frac{1}{c-1}\sum_{i=1}^{c}(N_i-\bar{N})^2}\ ,\ \bar{N} = \frac{1}{c}\sum_{i=1}^{c} N_i$$

By combining the above criteria we build the Quality Constraints System (QCS). The CRL_{max}, DB_{max}, MF_{min}, δ_{max} and the CLD_{max} must be given by the user with respect to the desired system performances.

$$QCS: \left\{\begin{array}{c} 1.\forall C_i, i \in [1,c]\ \frac{E_i}{I_i} \leq CRL_{max}; \\ 2.DB = \frac{1}{c}\sum_{i=1}^{c} max_{j\neq i}\left[\frac{diam(C_i)+diam(C_j)}{d(C_i,C_j)}\right] \wedge \\ (DB \leq DB_{max}) \\ 3.MF = \frac{\sum_i I_i}{\sum_i \frac{N_i(N_i-1)}{2}} - \frac{\sum_{i<j} E(C_i,C_j)}{\sum_{i<j} N_i N_j} \wedge \\ (MF \geq MF_{min}); \\ 4.Sh(f_i) = \frac{d(f_i,C_j)-d(f_i,C_i)}{max(d(f_i,C_j),d(f_i,C_i))} \wedge \\ ((1-Sh(f_i)) \leq \delta_{max}); \\ 5.CLD = \sqrt{\frac{1}{c-1}\sum_{i=1}^{c}(N_i-\bar{N})^2 \wedge} \\ (CLD \leq CLD_{max}); \end{array}\right\}$$

4.4 Cost Constraints

The *Cost Constraints System* (CCS) is build based on the system's cost criteria. Each ECU has a price and a performance description associated to it. We use the following constraints in order to build the CCS.

1. *The Price Constraint.* Given a network of function blocks F, a set of ECUs and a desired general cost DGC, then we have to distribute all the function set F over a number N_E of ECUs such that the total cost of these ECUs, P_{N_E} is smaller than DGC
2. The Bus Load Constraint (BLD) of the system must be lower than an imposed value, BLD_{max}. That is, we have to choose the ECUs on which we distribute the function blocks, such that the bus load of the system is never greater as the imposed value BLD_{max}.

By unifying the RCS with the QCS with the CCS and the TCS we derive the CSP associated to the system:
$CSP = RCS \cup QCS \cup CCS \cup TCS$.

Combining this four types of constraints restricts the possible set of configurations to an optimal solution.

4.5 Generative Deployment of Clusters

When the partitioning process is finished we start to deploy the clusters on the available ECUs. We use the generative constraints in order to choose the best available ECU that suits the requirements of the cluster.

We have different types of ECUs on which we must assign the clusters. We select the cluster that we want to deploy.The requirements of this cluster are generated by means of the generative constraints, e.g. memory, processing power, time to compute the assigned tasks. We choose from the set of ECUs the cheapest ECU and try to deploy the cluster. If, after replacing the meta variables with the concrete variables, an inconsistency happens between the resource provided by the ECU and the requirements of the cluster, we drop the selected type of ECU and chose another one. If the memory was not sufficient then we choose another type of ECU that provides the same processing power but enough extra memory to accommodate the requirements. We repeat this technique until all clusters are deployed. The case when a cluster can not be deployed, because there is no type of ECU to accommodate it, is eliminated in the cluster's building process. When generating clusters we generate them such that no cluster's requirements exceed the maximal resource capacity provided by the best type of ECU.

Using this technique we avoid searching for a best fit ECU. Instead of performing a (expensive) number of comparisons each time a cluster is deployed on a specific ECU. By means of generative constraints we focus exactly on the needed ECU. We do that by using the nogoods provided by the consistency checker. This obviously saves time and computational power.

5 Related Work and Conclusion

Constraints are a natural way of representing problems. They are often used in the area of configuration and reconfiguration of system. We use this straightforward and natural way of representing information and by means of a CSP solver we compute a valid solution which satisfies all the CSP criteria. Because of this approach, we do not have to generate all the cluster combinations but just wait for the first n; $n \geq 1$ solutions that the CSP solver delivers. Choosing a constraints solver proves to be a hard task to fulfill. As future work we also want to do an in depth comparison of the constraints solvers available on the market. Representing the partitioning problem as a CSP problem is a good extension to any other clustering algorithm. CSPs are also successfully applied in the area of configuration and reconfiguration of large software systems over a network, e.g. CAWICOMS which is presented in [15]. Other application areas for CSP representation are large business tasks planning.

This article proposes a constraint model for automated software deployment in embedded automotive systems. Unlike to specific algorithmic approaches, our purely model-based approach is applicable in an early stage of system development as there is no need for reference measurements on a prototypical implementation.

The idea behind our proposal is to use this algorithm in the early stages of the development. In order to fully optimize the solution we directly apply the generative deployment strategy on the given model. Hence we propose this algorithm as a methodology for deploying software components within an early state of the system's development.

Our novel CSP model notably extends previous models by incorporating (1) rigorous point-to-point timing requirements, (2) captures the optimal alignment of function blocks to synchronous as well as asynchronous ECU tasks by (3) relying the powerful technique of generative constraints. In terms of a simplified example, our article captures the critical issues in automating software deployment and thus further extends the practical application scope of (generative) CSPs.

REFERENCES

[1] Henrich Druck & Medien GmbH Challenges for the automotive supply chain,
Association of German Car Manufacturers (VDA) HAWK2015, Frankfurt am Main, 2003.

[2] E. Schoitsch Design for Safety AND Security of Complex Embedded Systems: A Unified Approach, *NATO Advanced Research Workshops, Cyberspace Security and Defense: Research Issues*, p. 161-174, Springer Dordrecht, Berlin, Heidelberg, New York.

[3] R. Henia, A. Hamann, M. Jersak, Razvan Racu, Kai Richter, Rolf Ernst System Level Performance Analysis - the SymTA/S Approach *IEE Proceedings Computers and Digital Techniques*, 152(2):148–166, March 2005

[4] S. Brummund, N. Kehl, P. Nenninger and U. Kiencke ISODATA Clustering for Optimized Software Allocation in Distributed Automotive Electronic Systems *SAE World Congress & Exhibition*,Detroit, MI, USA, Session: In-Vehicle Networks , 2006 .

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

[6] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On Tractable Queries and Constraints. In *Proc. 12th International Conference on Database and Expert Systems Applications DEXA 2001*, Florence, Italy, 1999.

[7] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[8] M. Yannakakis. Algorithms for acyclic database schemes. In C. Zaniolo and C. Delobel (Eds.), *Proc. of the International Conference on Very Large Data Bases (VLDB-81)*, Cannes, pp. 82-94, 1981.

[9] R. Debruyne and C. Bessiere From Restricted Path Consistency to Max-Restricted Path Consistency *SPrinciples and Practice of Constraint Programming*Berlin/Heidelberg ,pp. 312-326, 1997.

[10] D.Benavides, S. Segura, P. Trinidad and A. Ruiz-Cortes Using Java CSP Solvers in the Automated Analyses of the Feature Models *GTTSE*,Braga, Portugal, pp. 399-408, 2006.

[11] I. P. Gent, C. Jefferson and I. Miguel Minion: A Fast, Scalable, Constraints Solver*ECAI* Riva del Garda, Italy, 2006.

[12] M. Stumptner and F. Wotawa. Coupling CSP decomposition methods and diagnosis algorithms for tree-structured systems. In *Proc. 18^{th} International Joint Conf. on Artificial Intelligence*, pages 388–393, Acapulco, Mexico, 2003.

[13] F. Laburthe and N. Jussien CHOCO constraint programming system, *http://choco.sourceforge.net*, 2003-2006

[14] K. Kuchcinski Constraints-driven scheduling and resource assignment, *ACM Transaction on Desgin Atumation of Electronic Systems (TODAES)* ,8(3):355-383,July 2003

[15] L. Ardissono, A. Felfernig, G. Friedrich,D. Jannach, R. Schafer, M. Zanker A Framework for Rapid Development of Advanced Web-based Configurator Applications, *AI Magazine* , 24(3), 93-110, (2003).

[16] A. Felfernig, G. Friedrich, D. Jannach, M. Silaghi, and M. Zanker, Distributed Generative CSP Approach towards multi-site product configura-

tion, *Workshop on Immediate Applications of Constraint Programming (ACP)*,Cork, Ireland, 2003,100- 123

[17] Mihai Nica, Bernhard Peischl and Franz Wotawa, A Constraint Model for Automated Deployment of Automotive Control Software, *Proceedings of The 20th International Conference on Software Engineering and Knowledge Engineering*, Redwood City, San Francisco Bay, USA,2008.

A CSP based Distributed Product Configuration System

extended abstract

Fernando Eizaguirre and **Martín Zangitu** and
Josune de Sosa and **Karmele Intxausti**[1]

Abstract. This paper presents the current state of an ongoing research project aiming at developing a distributed configuration system for products structured as a hierarchy of nodes/subnodes distributed across internet. Each node maintains its own part of the product with its own parameters and constraints. The configuration problem is modelled as a distributed constraint satisfaction problem, CSP. As the user interacts with the configurator, the system allows selecting or unselecting product features, filtering out no longer consistent values and showing minimum and maximum cost of the product being configured. The paper describes the distributed CSP techniques used and gives a brief overview of the software prototype developed.

1 DISTRIBUTED PRODUCT CONFIGURATION

A distributed product can be described as a hierarchy of nodes and subnodes. Typically subnodes represent suppliers of components or parts of the product. In turn, each supplier can have its own second level suppliers, and so on. Therefore the product is distributed as a tree of nodes as it is shown in Figure 1. This example, representing the case of an elevator, has been used to test the system described in this paper.

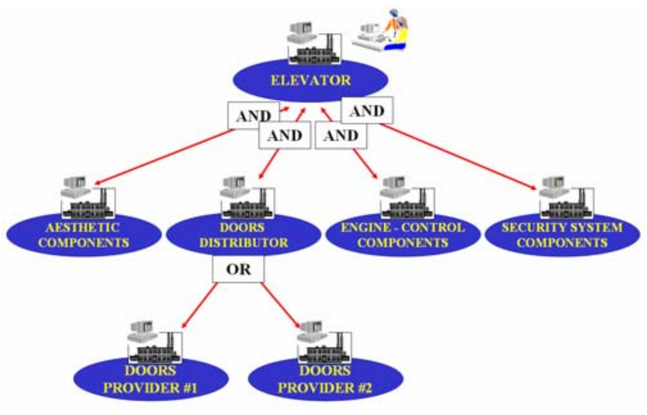


Figure 1. Distributed Product as a Node / SubNodes hierarchy.

Each node (i.e. a supplier) in the hierarchy exposes the parameters to configure its component so that any other node, higher in the hierarchy, can include the component parameters in its model; however, the constraints that restrict the valid combinations of the parameters defining the component are only known by the node and cannot be transmitted into a central node. In fact these constraints can be viewed as the private know-how of the company. Therefore, it is not possible to transmit all parameters and constraints into a central node to solve the problem with centralised CSP techniques

Each node in this *network* of configurators can play the role of being part of another product and, at the same time, can be itself a product. Therefore, the whole distributed configuration system can be seen as a hierarchy of distributed configurators in which each configurator can play the role of both a server (for client configurators higher in the hierarchy or even for human clients) and the role of a client (for configuration servers lower in the hierarchy). The whole network of configurators follows a client/server paradigm.

2 REQUIREMENTS OF A CONFIGURATOR

Requirements of configurators have been thoroughly identified and analysed in the literature and can be studied from different points of view. In [1] requirements are identified and discussed about the need of common knowledge representation among nodes, the management and maintenance of the product model, the need of adaptable and dynamic generated interfaces for users of different skill levels, distributed reasoning, etc. In [2] the authors describe a framework for the management of personalised configuration in business oriented domains that allows the system to dynamically generate personalised user interfaces. In [3] requirements as consistency, completeness, optimality and solution finding are discussed. In the research project described in this paper, due to limited scope and funding of the project, the authors focus its interest just in the distributed reasoning aspects. Apart from base requirements, such as consistency and automatic completion of solutions, the project focuses also in developing algorithms to satisfy the following requirements:

- **Selection of inconsistent values (correction of previous decisions)** – As the user interacts with the configurator, in some situations, the user may decide that he wants to select a given value marked as inconsistent by the configurator as a

[1] Ikerlan Technological Research Centre, Spain, email: feizaguirre@ikerlan.es

result of previous decision. In this case, the configurator should be able to propose to the user a list of previous decisions that should be changed so that the current inconsistent value he wants to select be consistent and eligible.

- **Minimum and Maximum cost** - After each interaction of the user, the configurator must calculate (in background) the solution with the minimum cost and the solution with the maximum cost compatibles with choices so far.
- **Real Time** - The above mentioned requirements must be satisfied in real-time (few seconds for human client).

3 REVIEW OF DISTRIBUTED ALGORITHMS

In recent years algorithms for distributed CSP have been proposed (see [4], [5]). Almost all the algorithms are based on a set of agents that cooperate to find a global solution. Algorithms can be basically classified along two axes: (a) number of variables managed by each agent and (b) synchronous / asynchronous communication among agents.

Most of the algorithms proposed in the literature, are based on agents that manage just one variable and therefore are not suitable for a model in which each node manages a set of variables and not just one variable. Obviously each node could be implemented a set of agents managing just one variable, but this implementation would not be efficient. In addition to this, these algorithms are designed to manage binary constraints, but unfortunately, in configuration problems, n-ary constraints are much more frequent.

In the project CAWICOMS (see [1], [6]) a hierarchy of configurators working in a client/server mode is developed in which each configurator manages its own set of variables and constraints. One of the algorithms developed in this project, a *synchronous configuration of subassemblies* is based in an extension of the forward checking mechanism (invoking synchronously subnodes to check/complete a partial solution to the subassembly when necessary) and is transparent to the search algorithm.

3.1 Synchronous Versus Asynchronous

Many of the distributed algorithms are based on an asynchronous communication allowing agents to work in parallel. To guarantee a global consistency (depending on the algorithm) each agent maintains a list of the unsuccessful partial solutions tried so far (list of nogoods) to keep track of solutions that are not globally consistent, avoiding repeating unsuccessful assignations. The idea behind asynchronous algorithms is that agents work in parallel so that efficiency is increased.

Unfortunately, as it is noted in [7], many of the studies about asynchronous algorithms have assumed instantaneous messages or have underestimated the impact of message delays. In a realistic scenario communication delays are not negligible at all, in fact in the time needed to propagate a message, a given agent node can do a lot of work in its local constraint network. This means that agents, most of the time, work with inconsistent information wasting the theoretical advantage over synchronous algorithms.

The paper [7] shows an improved version of a simple synchronous backtracking (with backjumping) and concludes that, with message delays, it performs equally well as asynchronous search algorithms in terms of computational effort. In addition to this, the paper also concludes that the synchronous backtracking loads the network with only one message at a certain time and its performance is stable for a variety of communication qualities.

In [8] the same authors present an interesting asynchronous algorithm in which multiple and dynamically created search process work in parallel in non intersecting parts of the problem. Each individual search process is a synchronous search algorithm working in its own part of the domain problem. This is a very interesting idea that couples the advantage of synchronous algorithms (robust with message delays) and asynchronous algorithms (all agents working in parallel avoiding idle agents).

3.2 Constraint Compilation

There exist approaches to configuration oriented to ensuring global consistency in real-time (one the key requirements of a configurator), for example, such as those based on compilation of constraint networks (see [11]). Moreover, there are commercial companies selling configurators based on compilation of constraints (see [12] and [13]). However these techniques are not oriented to distributed problems and are not well suited to changing conditions of supply-chain networks. In addition to this, consistency of cost variables (maintaining minimum and maximum cost of the product as it is configured), has a high impact in these techniques that may become not viable.

4 DISTRIBUTED CSP MODEL ADOPTED

This section describes the distributed CSP model adopted in the project. As shown in Figure 2, each node in the hierarchy holds its own constraints network with its own variables and constraints representing the product/component/part sold by the company.

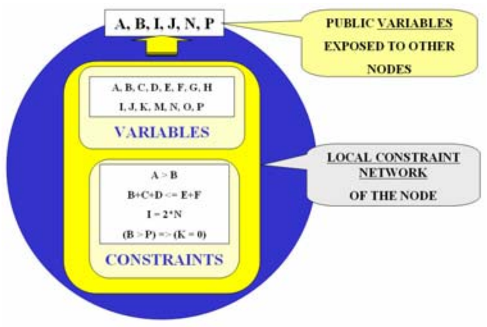


Figure 2. Public variables exposed by each node.

A subset of the variables of the local CSP of each node, represent the configuration parameters of the component/part. This set of variables is public and can be accessed by other nodes higher in the hierarchy.

Consistency among nodes and subnodes follows a similar approach to the one described in [6] and is basically an extended version of a forward checking (that implements a synchronous forward checking to the subnodes) and is transparent for the search algorithms. In the father node, consistency between the public variables of a given subnode and the corresponding variables in the father node (see Figure 3) is kept by a "*remote equality constraint*" that keeps these variables arc-consistent

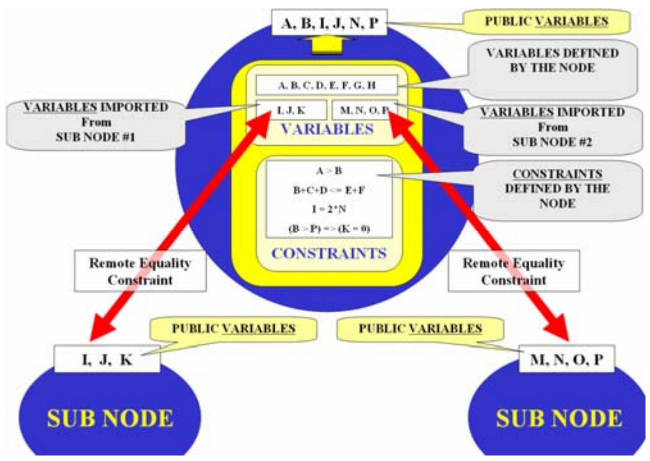


Figure 3. Links between CSPs of Nodes and SubNodes.

This constraint (that implements an AC-4 consistency, see [10]) works as follows:

- Whenever the domain of any of the imported variables in the father node changes, the constraint is invoked. The constraint collects the current state of the domains of the imported variables in the father node and sends these domains to the configurator of the subnode for checking consistency. The calling configurator, the father node, waits synchronously for the subnode's reply.
- When the subnode's configurator receives the request, it checks with its local constraint network if the domains received in the request are consistent, pruning domains until they are arc-consistent. If any of the domains is empty, the subnode's configurator answers back with a message asking for backtracking. Otherwise, the subnode's configurator answers back with a message containing the new filtered and arc consistent domains. After answering to the request, the subnode's configurator remains waiting for new requests.
- When the calling constraint in the father node receives the answer with the new arc-consistent domains, immediately applies the domains to the imported variables and the search process continues.

For the communication mechanism, two alternatives were tested for implementation, the use of XML over SOAP, as in CAWICOMS project, and the use of Sockets over TCP/IP. The former is a secure communication mechanism but unfortunately, even in a local network, it performed quite slow for the extended forward checking implemented in this system (in fact XML over SOAP can achieve just a few transactions per second). The latter, sockets over TCP/IP, has some security concerns (server needs an open port and firewalls must be configured to not blocking connections) but is very fast. In a local area network, inside a company, a socket can perform thousands of transactions per second. Within a virtual private network with an optical fiber, sockets can perform several hundreds of transactions per second. However security concerns and firewalls are issues to be solved.

5 SEARCH ALGORITHMS

The distributed CSP model just described and the search algorithms are implemented on top of SICStus Prolog clpFD © (see [9] for details). Algorithms (implemented on top of the *labeling* predicate of SICStus for which the extended forward checking works transparently) are basically a synchronous backtracking with some improvements to minimize the network load. These algorithms have been tested with the example of the elevator, that is a small size model with 7 nodes, and about 5 variables per node (apart from imported variables from subnodes) with a domain size about 10 values per variable, and about 15 simple constraints per node.

Consistency will not be discussed here because it is considered that arc-consistency level, provided by the SICStus clpFD engine, is a sufficient condition for an acceptable configurator, even if it does not guarantee that all the values of the current domains of the variables are part of a global valid solution (the cost of ensuring that condition would be prohibitive because in fact it is equivalent to find one solution for each possible value of all variables, unless constraint compilation techniques were used, not the case here). In practice, it is the opinion of the authors that arc-consistency may be sufficient for most of users.

Regarding, automatic completion of solutions, it is implemented by using *labeling* algorithm of SICStus and simply consists in finding the first valid solution compatible with choices made so far.

5.1 Selection of inconsistent values

As the user interacts with the configurator making decisions (selecting or eliminating values of variables) the arc-consistency process may eliminate values of other variables. At a given point, the user may find that, for a variable Var_J, he wants to select a value Val_J that has already been eliminated by the arc-consistency process. In spite of being marked as an invalid value the user may ask to the system to tell which previous decisions need to be undone in order to be able to select value Val_J.

The algorithm implemented to meet this requirement is quite simple. For each previous decision D_i made by the user (variable Var_i taking the value Val_i), the configurator (starting from the original problem without user's decisions) defines a variable called $KeepD_i$ (domain boolean True/False) and post the reified constraint that states ($KeepD_i$ => (Var_i = Val_i)). Along with these constraints, the configurator post the constraint (Var_J = Val_J) that is the value the user wants to select. Next the configurator tries to find a solution (assignment to all $KeepD_i$ variables) maximising the number of variables $KeepD_i$ taking the value True. Finally the configurator proposes to the user the previous decisions that should be changed ($KeepD_i$ assigned False) to enable the value he wants to select.

An obvious improvement to this algorithm is to assign a weight (an importance) to each previous decision (for example, first decisions are probably more important than later decisions) and find the solution that minimises the total weight of decisions to be changed.

5.2 Minimum and maximum cost

After each interaction of the user, the configurator simply tries to find the solution (in background without changing current state of the configuration process), consistent with user's choices so far, that (a) minimizes the cost variable (only one cost variables is allowed) and the solution that (b) maximizes the cost variable, reporting those values in the interface. For a small size problem as the one described in this paper, almost real-time (few seconds) is achieved in all the cases tested. The algorithm used is the *labeling* (with maximize or minimize option) provided by SICStus Prolog clpFD that uses a branch-and-bound algorithm.

6 SPEEDING UP THE SEARCH PROCESS

6.1 List of nogoods and "goods"

Each remote equality constraint stores the list of all past requests/answers (in a given interaction of the user) to subnodes (not only the *nogoods* are stored but also the "*goods"*). This way, before requesting something to a configurator of a subnode, the constraint consults its list to check if that request was done before, avoiding repeating the same work. The price to be paid by this mechanism is memory consumption but the gain in speed is very important because, due to well know problems of backtracking (for example trashing) the same request may be done many times to the subnodes.

6.2 Backjumping

Search algorithms were improved by adding a graph-directed backjumping algorithm (see [14] for details of GBJ algorithm) rewriting the *labeling* algorithm of SICStus Prolog clpFD but improvement was not clear for the elevator model (to be checked with other models). It was tried to implement the Conflict-Directed Backjumping (see [15] for details) but unfortunately this revealed quite complex due to the fact that SICStus Prolog clpFD engine does not provide predicates to determine easily the cause of failures when a backtrack is produced.

7 SOFTWARE PROTOTYPE

To test the techniques described in this paper, a software prototype has been implemented. This prototype includes (a) an editor allowing the user defining the distributed product model, (b) a module that generates automatically the SICStus Prolog clpFD code to be deployed in each node, and (c) a module that generates automatically a simple web-based interface (just to test the system) of the root node of the hierarchy.

8 CONCLUSION AND NEXT STEPS

This paper has described the current results of an ongoing research project aiming at developing a software prototype for distributed product configuration system. The paper, by no means, aimed at developing new techniques or algorithms in constraint satisfaction, but consisted of a practical application of a variety of known techniques in constraints satisfaction on top of a commercial solver. Next steps in the project include (a) implementing a concurrent search algorithm based on the ideas explained in [8] because concurrent search over non intersecting parts of the domain is a way of taking advantage of parallelism while keeping algorithms synchronous, and (b) testing the system performance with problems of different size and complexity.

ACKNOWLEDGEMENTS

The authors want to thank the Basque Government, for funding the project, and the referees for their comments which helped improve this paper.

REFERENCES

[1] Ardissono, L. and Felfernig, A. and Friedrich,G. and Goy, A. and Jannach, D. and Petrone, G. and Schäfer,R. and Zanker, M. A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems, AI Magazine**,** Volume 24 , Issue 3, September 2003, pp 93 – 108, ISSN:0738-4602

[2] Ardissono L., Felfernig A., Friedrich G., Goy A., Jannach D., Meyer M., Petrone G., Schäfer R., Schütz, W., Zanker M.: Personalizing on-line configuration of products and services. Proceedings of the 15th European Conference on Artificial Intelligence, pp. 225-229, Lyon, France, IOS Press 2002

[3] Haag, A. Sales configuration in business processes. IEEE Intelligent Systems & Their Applications, Vol 13, Issue 4, pp 78-85, 1998

[4] Makoto Yokoo. Distributed constraint satisfaction: foundations of cooperation in multi-agent systems. Springer-Verlag, 2001. ISBN 3-540-67596-5

[5] C. Bessiere and A. Maestre and P. Messeguer. Distributed Dynamic Backtracking. Proc. Workshop on Distributed Constraints, IJCAI-01, Seattle, 2001.

[6] Ardissono, L. and Felfernig, A. and Friedrich,G. and Jannach, D. and Schäfer,R. and Zanker, M. Framework for Rapid Development of Advanced Web-based Configurator Applications. In proceedings of 15th Conf. ECAI 2002, Lyon

[7] R. Zivan and A. Meisels. Synchronous vs asynchronous search on DisCSPs. Proceedings of the 1st European Workshop on Multi Agent System, EUMAS, Oxford, December 2003

[8] R. Zivan and A. Meisels. Concurrent search for distributed CSPs. Artificial Intelligence, Volume 170. Issue 4, pp 440-461, April 2006.

[9] Mats Carlsson and Greger Ottosson and Bjorn Carlson. An open-ended finite domain constraint solver. In H. Glaser, P. Hartel, and H. Kucken, editors, PLILP'97, International Symposium on Programming Languages: Implementations, Logics, and Programming, number 1292 in LNCS, Southampton, September 1997

[10] R. Mohr and T.C. Henderson. Arc and path consistency revisited. Artificial Intelligence, 28: 225-233, 1986

[11] Weigel, R., Faltings, B.: Compiling constraint satisfaction problems. Artificial Intelligence 115 (1999) 257—289

[12] T. Hadzic and S. Subbarayan and R. M. Jensen and H. R. Andersen and J. Møller and H. Hulgaard . Fast Backtrack-Free Product Configuration Using a Precompiled Solution Space Representation. In Proc. of the International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, 28-29 June 2004, Copenhagen, Denmark.

[13] Tacton Configurator. www.tacton.com

[14] Dechter, R. and Frost D. Backjump-based Backtracking for Constraint Satisfaction Problems Artificial Intelligence,. 136:147–188, 2002

[15] Prosser, P. Hybrid algorithms for the constraint satisfaction problem. Computational Intelligence, 9(3):268-299.1993

Constraint-based personalized bundling of products and services

Markus Zanker, Markus Aschinger and Markus Jessenitschnig [1]

Abstract. The composition of product bundles such as tourism packages, financial services or compatible skin care products is a synthesis task that requires the support of a knowledgeable information system. We present a constraint-based Web configurator capable of solving such tasks in e-commerce environments. Our contribution lies in hybridizing a knowledge-based configuration approach with collaborative methods from the domain of recommender systems in order to exploit user preferences to guide the solving process through large product spaces. The system is implemented on the basis of a service-oriented architecture and supports a model-driven approach for knowledge acquisition and maintenance. An evaluation of the system suggests that it can solve realistic problem instances within an acceptable computation time.

1 Introduction

In many e-commerce situations consumers are not looking for a single product item but rather require a set of several different products. For instance on e-tourism platforms online users may either selectively combine different items, such as accommodation, travel services, events or sights, or they might choose from an array of pre-configured packages. While bundling different items on their own users are performing a synthesis task comparable to configuration.

Mittal and Frayman [11] defined configuration as a special type of design activity, with the key feature that the artifact being designed is assembled from a set of pre-defined components. When bundling different products together, product categories represent these pre-defined components. Additional knowledge is required that states which combinations of components are allowed and which restrictions need to be observed. For instance, proposed leisure activities should be acceptably close to the guest's accommodation or recommended sights need to be appropriate for children if the user represents a family. Nevertheless, the problem of computing product bundles that are compatible with a set of domain constraints differs from traditional configuration domains in the sense that fewer restrictions apply. For instance a car configurator must compute a valid vehicle variant satisfying the user's requirements and all applicable commercial and technical restrictions derived from the manufacturer's marketing and engineering policies. In contrast, few strict limitations apply to a product bundle, because it represents an intangible composition of products rather than a new artifact. As a consequence, an additional order of magnitude of component combinations are possible and the question of finding an optimal configuration becomes crucial.

Optimality can be either interpreted from the provider perspective, e.g. the configuration solution with the highest profit margin, or from the customer perspective. In the latter case, the system should propose the product bundle that best fits the customer's requirements and preferences. Typically, recommender systems are employed to derive a ranked list of product instances for an abstract goal such as maximizing user's utility or online conversion rates [1]. The most commonly used recommendation technique is collaborative filtering, which exploits clusters of past users with similar interests (peer users) to propose products that were well liked by these peers [15]. We exploit this type of recommender system to order personalized preferences for each type of product in our configuration problem. Our contribution thus lies in integrating soft preference information obtained from recommender systems (based on for instance the collaborative filtering paradigm) into a constraint-based configuration approach allowing user preferences to guide the system towards finding optimal product bundles.

In contrast to the work of Ardissono et al. [2] and of Pu and Faltings [14], we do not require explicit preference elicitation via questioning or example-critiquing, but depend on the underlying recommendation paradigm of community knowledge and past transaction data.

The paper is structured as follows: in Section 2 we present an extensive survey of related work before introducing a motivating example in Section 3. Furthermore, we elaborate on the system's development in Section 4 and conclude by summarizing practical experiences and results.

2 Related Work

Configuration systems are one of the most successful applications of AI-techniques. In industrial environments, they support the configuration of complex products and services and, compared to manual processes, help to reduce error rates and increase throughput. Depending on the underlying knowledge-representation mechanism, a rule-based, model-based or case-based framework may be employed for product configuration [17]. Configurators that utilize the constraint satisfaction problem (CSP) paradigm are within the family of model-based approaches [6, 10] and include an explicit knowledge base that is distinct from the system's problem solving strategy. In technical domains such as telephone switching systems, large problem instances with tens of thousands of components exist. Efficient strategies for solving problems exploit the functional decomposition of the product structure to determine valid interconnections of the different components [6]. Pure sales configuration systems, such as online car or pc configuration systems[2], are much simpler from a computational point of view. They allow their users to explore the

[1] University Klagenfurt, Universitätsstrasse 65-67, 9020 Klagenfurt, Austria E-mail: {firstname.lastname}@uni-klu.ac.at

[2] For instance, see http://www.bmw.com or http://store.apple.com

variant space of different options and add-ons and ensure that users place orders that are technically feasible and correctly priced. However, these systems are typically not personalized, i.e. they do not adapt their behavior according to their current user.

The CAWICOMS project was among the first to address the issue of personalization in configuration systems [2], developing a framework for personalized, distributed Web-based configurators. The system's dynamic user interfaces adapt their interaction style according to abstract user properties such as experience level or needs. The system decides on the questioning style (e.g. asking for abstract product properties or detailed technical parameters) and computes personalized default values if the user's assumed expertise is insufficient. Pu and Faltings [14] present a decision framework based on constraint programming and demonstrate the suitability of soft constraints for supporting preference models. Their work concentrates on explicitly stated user preferences and presents an example critiquing interaction model to elicit tradeoff decisions from users. Given a specific product instance users may provide critique on one product property and specify which other properties they would be willing to compromise on. Soft constraints with priority values are revised in such an interaction scenario and guide the solution search.

In contrast to the work in [2, 14], we do not solely rely on explicitly stated user feedback, but integrate a recommender system into the configuration process to include assumed user preferences.

Recommender systems constitute a base technology for personalized interaction and individualized product propositions in electronic commerce [1]. However, they do not support synthesis tasks like configuration. Given sets of items and users, recommender systems compute for each single user an individualized list of ranked items according to an abstract goal such as buyer interest or the likelihood of a sale [1]. Burke [5] differentiates between five different recommendation paradigms: *collaborative*, *demographic* and *content-based* filtering as well as *knowledge* and *utility-based* recommendation. Collaborative filtering is the most well known technique that utilizes clusters of users that showed similar preferences in the past to provide recommendations to users in the same neighborhood [15, 13].

Content-based filtering records the items that were liked by the user in the past and proposes similar ones. Successful application domains are, for instance, the suggestion of news or Web documents in general, where the system learns user preferences in the form of vectors of term categories [3]. Demographic filtering builds upon the assumption that users with similar social, religious or cultural background share similar views and tastes. Knowledge and utility-based methods rely on a domain model of assumed user preferences that is developed by a human expert. Jannach [7] developed a sales advisory system that maps explicitly stated abstract user requirements onto product characteristics and computes the set of best matching items. Case-based recommender systems exploit former successful user interactions denominated as cases. When interacting with a new user, the system retrieves and revises stored cases in order to make a proposition. Subsequently, a human expert is required to define efficient similarity measures for case retrieval [12]. Burke [4] and Ricci [16] have carried out extensive research within this field and have developed several successful recommendation systems.

We chose to integrate a polymorphic recommendation service into our configurator that can be instantiated with an arbitrary recommendation paradigm [23]. Details of this process are provided in Section 4.

Preference-based searches require a more interactive and dynamic approach for the personalized retrieval of items. Rather than relying on a static preference model, they build and revise the preference model of the specific user during interaction. One of the first applications of interactive assessment of user preferences was the Automated Travel Assistant [9]. Further work on interactive preference elicitation has been conducted recently [18, 19, 21] and [22] includes an extensive overview.

The work on preference-based search is orthogonal to our contribution as we primarily focus on computing bundles of recommendations. Our implementation supports interactivity between the system and the user during exploration of the search space. Therefore, additional preference constraints can be added and revised during each round of interaction (see Subsection 4.5).

3 Motivating example

To describe our approach, we start by giving a motivating example. Figure 1 depicts a service configuration scenario from the e-tourism domain. The user model states a set of specific requirements for *John*, such as that he is interested in a travel package to the city of Innsbruck or that the solution should be appropriate for a family with children (see dotted arrows marked with 1). In addition, contextual parameters such as the weather situation or the current season are represented within the system context. Concrete product instances and their evaluations are part of the product model, e.g. sights or restaurants. The dotted arrows exemplify some constraints of the configuration knowledge base like *the location of the sight/restaurant/event and the location of the accommodation should be the same* (cmp. mark nr. 2) or *If the weather outlook is rainy propose an indoor event* (cmp. mark nr. 3).

Additional preference information is included in the configuration knowledge base by integrating recommendation services for each product class. For instance, when collaborative filtering recommends items from the product class *Event*, transaction records of other users and the interaction history of *John* are also exploited. Consequently, higher ranked recommendation items are more likely to be included in the configuration solution.

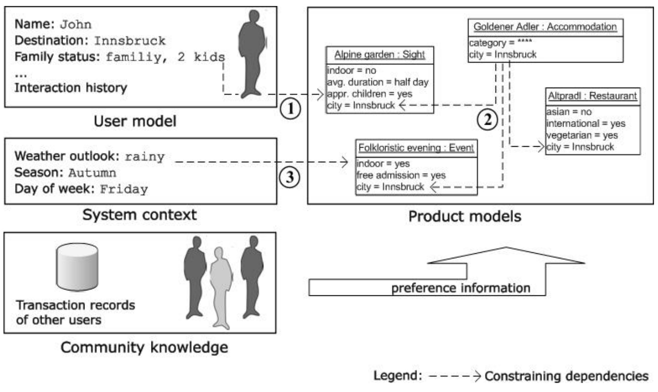


Figure 1. Example scenario

Informally, given a user model, a system context and a set of constraints, the task of the system is to find a set of products from different categories that is optimal with respect to the preference information derived from external recommender systems. In the following we will detail the development of the system.

4 Development

While designing the system we decided to use the constraint programming paradigm for knowledge representation and problem solving - comparable to most of the configuration systems referenced in Section 2. The Java *Choco* Open Source constraint library [8] forms the basis of our implementation. The following subsections describe the constraint-representation of the domain model, various aspects of knowledge acquisition as well as our system architecture.

4.1 Architecture

Figure 2 illustrates the system's architecture. It consists of a configuration service component and several recommender services for delivering personalized instance rankings from a given class of products. The implementation utilizes a service-oriented architecture that supports communication via Web services, a php-API and a Java-API, enabling flexible integration with a wide range of Web applications, distributed deployment scenarios and ensures that the system can be extended to include additional recommendation services. The latter requires sharing the identities of users and product instances as well as the semantics of user and product characteristics among all components. This is realized by a central user and product model repository that also offers service APIs.

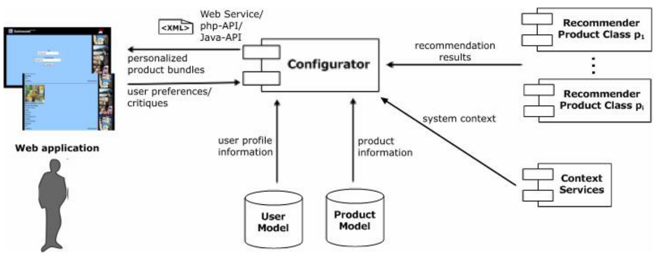


Figure 2. System architecture

The user interacts with a Web application that itself requests personalized product bundles from the configurator via the service-API. The evaluation of contextual parameters can be requested using the same means of communication. We have implemented variants of collaborative and content-based filtering recommenders as well as utility and knowledge-based ones as sketched in [23]. A more detailed evaluation of different recommendation strategies on commercial data was performed in [24]. Next, we will examine the domain model for the configuration service itself.

4.2 Model representation

As stated above, the constraint satisfaction paradigm is employed for knowledge representation. A Constraint Satisfaction Problem (CSP) is defined as follows [20]:

A CSP is defined by a tuple $\langle X, D, C\rangle$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{d_1, \ldots, d_n\}$ a set of corresponding variable domains and $C = \{c_1, \ldots, c_m\}$ a set of constraints.

Each variable x_i may only be assigned a value $v \in d_i$ from its domain. A constraint c_j further restricts the valid assignments within a set of variables. For each partial value assignment to variables it is possible to determine if a constraint has been violated or not. In addition, all constraints $c_j \in C$ are defined to be either *hard* ($c_j \in C_{hard}$) or *soft* ($c_j \in C_{soft}$), where $C = C_{hard} \cup C_{soft}$ and $C_{hard} \cap C_{soft} = \emptyset$. Soft constraints may also be violated by variable assignments. Each one is typically associated with a penalty value and the sum of penalty values of violated constraints has to be minimized when looking for an optimal solution. For further details and definitions of CSPs we refer the reader to [20].

Next, based on this formalization we present our domain model. It consists of a tuple $\langle X_{\{P,\, UM,\, Cx,\, PM\}}, D_{\{P,PM\}}, C_{\{hard,soft\}}\rangle$, where:

- $X = X_{UM} \cup X_{Cx} \cup X_P \cup X_{PM}$ the set of variables subdivided into several disjoint subsets as explained below,
- $D = D_P \cup D_{PM}$ the sets of corresponding domains for the product classes and their properties,
- $C = C_{hard} \cup C_{soft}$ the set of constraints subdivided into hard and soft constraints,
- $X_{UM} = \{u_1, \ldots, u_j\}$ a set of variables from the user model,
- $X_{Cx} = \{cx_1, \ldots, cx_k\}$ a set of variables modeling the system context,
- $X_P = \{p_1, \ldots, p_i\}$ a set of index variables representing the product classes, each associated with a recommendation service that delivers a personalized item ranking upon request,
- $weight(p_i)$ the relative weight of product class p_i used in the overall optimization function,
- $X_{PM} = \{p_1.x_1, \ldots, p_1.x_m, \ldots, p_i.x_1, \ldots, p_i.x_n\}$ a set of variables modeling product properties, where $p.x$ denotes the product property x of product category p and $p[j].x$ the concrete evaluation of x for product instance j,
- $D_P = \{d_1, \ldots, d_i\}$ a set of corresponding domains for the product classes,
- $D_{PM} = \{p_1.d_1, \ldots, p_1.d_m, \ldots, p_i.d_1, \ldots, p_i.d_n\}$ a set of corresponding domains for product properties,
- $C_{hard} = \{c_1, \ldots, c_p\}$ a set of hard constraints on variables in X,
- $C_{soft} = \{c_1, \ldots, c_q\}$ a set of soft constraints on variables in X and finally
- $pen(c_j)$ the penalty value for relaxing soft constraint c_j.

This domain model is defined and maintained solely by domain experts in order to reduce the traditional knowledge acquisition bottleneck as outlined in the next subsection.

4.3 Model definition and CSP generation

Model definition and maintenance is supported by a modular editor environment based on the Eclipse RCP (Rich-Client Platform) technology[3]. Figure 3 gives an overview of the interaction with the knowledge acquisition workbench. First, the relevant characteristics for configuring a bundle are retrieved from the user model repository. Second, additional external services providing contextual data such as the current season of the year or weather information are selected. In a third step, the set of product classes P and their associated recommender services are integrated. For each class of products relevant properties are selected from the underlying repository. Finally, hard and soft constraints are defined using a context-sensitive editor.

In Figure 4 depicts the complete process. This can be separated into a design phase, where the model is defined and maintained, and an execution phase. During the latter, the configurator is invoked for a specific user u. First, product rankings are retrieved from recommendation services and then corresponding product characteristics

[3] See http://www.eclipse.org/rcp for reference.

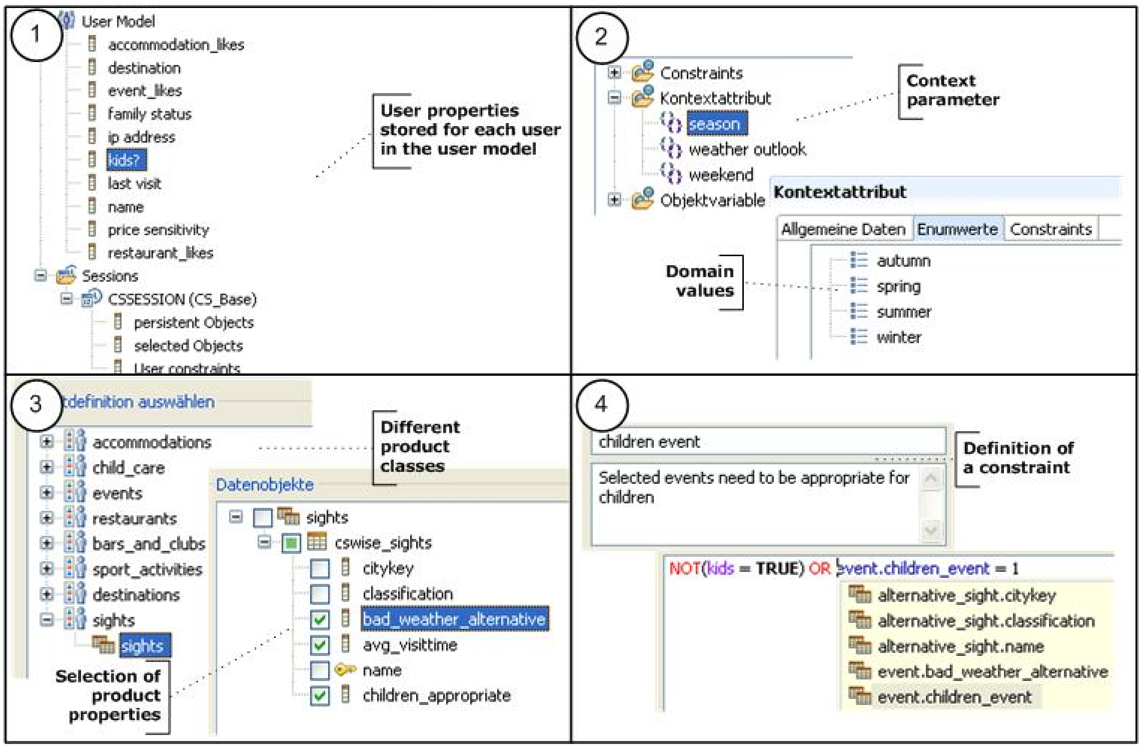


Figure 3. Knowledge acquisition workbench

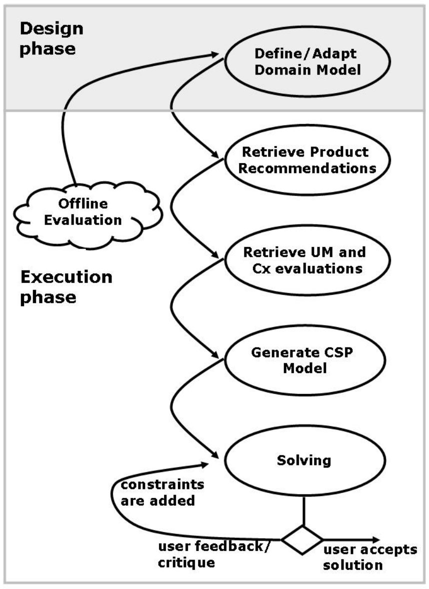


Figure 4. Design and Execution phase

are requested from the product model repository. In the next step, all variables in $X_{UM} \cup X_{Cx}$ are assigned values by the particular user model repository and the context services.

Then a CSP model is generated on the fly and subjected to the following transformation steps:

- Create all variables in $X_{UM} \cup X_{Cx}$ in the CSP model and assign them their respective evaluations.
- For each index variable $p_i \in P$ we create the related domain $d_i = \{1, \ldots, n\}$, where n is the number of recommendations received for product class i. Product instance $p_i[1]$ denotes the highest ranked recommendation and $p_i[n]$ the lowest ranked respectively. The index represents the preference information about the instances of a product class. If two product instances fulfill all constraints then the one with the lower index value should be part of the recommended bundle.
- Create all variables in X_{PM} and assign them domains as follows: $\forall p_i \in X_P, \forall p_i.x_j \in X_{PM}, \forall u \in d_i \quad p_i[u].x_j \in p_i.d_j$, i.e. for a given product property x_j from product category p_i all characteristics of recommended instances need to be in the domain $p_i.d_j$.
- Furthermore, integrity constraints are required to ensure that the value assigned to the index variable p_i of product class i is consistent with the values assigned to its product properties $p_i.x$: $\forall p_i \in X_P, \forall p_i.x_j \in X_{PM}, \forall u \in d_i \quad p_i = u \rightarrow p_i.x_j = p_i[u].x_j$, i.e. when choosing the u-th instance of product class i ($p_i = u$), the related product property x_j is assigned the value of the u-th instance. Hence a complex product representation is supported, although the *Choco* CSP formalism does not support object oriented notation.
- Insert all domain constraints from $C_{hard} \cup C_{soft}$.
- For each soft constraint $c \in C_{soft}$, create a variable $c.pen$ that

holds the penalty value $pen(c)$ if c is violated or 0 otherwise.

- Create a resource variable res and a constraint defining res as the weighted sum of all variables holding penalty values for soft constraints and all weighted index variables. The tradeoff factor between soft constraints and product class indexes as well as all weights can be adapted via the knowledge acquisition workbench.

4.4 CSP solving

Once the CSP model is generated, the *Choco* solver is invoked. Its optimization functionality aims to find an assignment to all variables in the CSP model that does not violate any hard constraint and minimizes the resource variable res. We extended the branch and bound optimization algorithm [8] to compute the top n solution tuples instead of solely a single product bundle. The solver is capable of following two different strategies for computing n product bundles, namely *1-different* and *all-different*. The *1-different* strategy ensures that each tuple of product instances in the set of n solutions contains at least one product instance that is different to all other solution bundles. In contrast, the *all-different* variant guarantees that the intersection between two product bundles from the set of solutions is empty, i.e. a product instance may only be part of at most one solution tuple. In section 5 we present the computation times for CSP generation and solving.

4.5 Interactivity

The system also supports a constant user interaction throughout the configuration process. As described in Figure 4, the user either accepts a proposed solution or requests a new configuration. In the latter case she or he may provide some feedback by explicitly accepting or rejecting some components of the proposed bundle which results in explicit equality or inequality constraints being added. The rich critiquing interface that accepts user feedback such as *'no traditional restaurants'* or *'the hotel should be located closer to the town center'* supports an incremental preference elicitation strategy comparable to [22]. Preferences can be expressed by arbitrary hard and soft constraints as defined in our domain model. Thus, in each round of interaction additional constraints will be added to the CSP model. Furthermore, for the sake of completeness, interaction sessions can be restored and resumed at a later point of time.

5 Evaluation

Based on our e-tourism application domain we developed an example scenario consisting of 5 product classes with a total of 30 different product properties. Their domains are strings, bounded integers or boolean values. We defined a total of 23 representative domain constraints (13 hard and 10 soft constraints) as configuration knowledge.

We created 5 different CSP models (denoted M1 to M5) with between 5 and 100 requested product instances. Details of the problem sizes and computation times for 'on-the-fly' generation of CSP models are given in Table 1. Clearly, the number of variables does not depend on the number of recommendations and is constant for all models. However, the average size of variable domains increases from M1 to M5 due to the higher amount of product instances per product class. The number of constraints depends mainly on the aforementioned integrity constraints on model product instances. Therefore, M5 contains ten times the number of constraints found in M1. As can be seen in Table 1, the times for generating even the large M5 model are acceptable for interactive applications.

Model	Number of Recs.	Number of Vars	Average Domain Size	Number of Constraints	Generation time in ms
M1	5	58	7,45	206	10
M2	10	58	8,73	374	20
M3	30	58	13,55	1010	60
M4	50	58	16,5	1355	95
M5	100	58	23,23	2093	135

Table 1. Model sizes used in evaluation

In order to evaluate the performance of the system experiments were conducted using a computer containing a standard Pentium 4 3GHz processor. The time measurements for the solving steps are depicted in Figure 5. A series of 100 test runs (representing 100 different users) were executed for each model to obtain an average computation time. We evaluated both strategies, computing a set of top n solution tuples, where n was varied between 1 and 10. The solving times (y-axis) for the *all-different* strategy are presented using a logarithmic scale. Moreover, graphs for M3 and M4 are omitted for readability. The *1-different* strategy is significantly less complex and therefore solve times did not exceed 50 ms. In contrast, *all-different* requires significantly longer.

Nevertheless, performance is still satisfactory. The highest value for model M5 (for 10 solutions) was computed in about 1.5 seconds. Although typical e-commerce situations require at most 10 different product bundle suggestions, requests for as many as 100 solution tuples can still be computed within a acceptable time period. For instance, when using the *1-different* solution strategy the solver required around 220 ms to calculate the top 100 solutions for model M5. In the case of the *all-different* strategy, a maximum of 15 solutions could be found in this problem instance. The associated calculations required 6 seconds which indicates that this would be be the likely bottleneck of an interactive system.

Nevertheless, our results indicate that the integration of different recommender systems into a configurator is efficient enough to solve standard online product bundling tasks. Further experiments in different example domains will be conducted as future work.

6 Conclusion

We presented the development of a generic Web configurator that combines recommendation functionality with a constraint solver. Our research contribution lies in the system's novel strategy of personalizing configuration results of product bundles. The system observes on the one hand explicit domain restrictions and on the other hand user preferences deriving from recommender systems. A test application was developed within the scope of an industrial research project in the e-tourism domain which subsequently showed that the system performs acceptably for realistic problem sizes.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin, 'Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering*, **17(6)**, (2005).

[2] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, G. Petrone, R. Schäfer, and M. Zanker, 'A framework for the development of personalized, distributed web-based configuration systems', *AI Magazine*, **24**(3), 93–108, (2003).

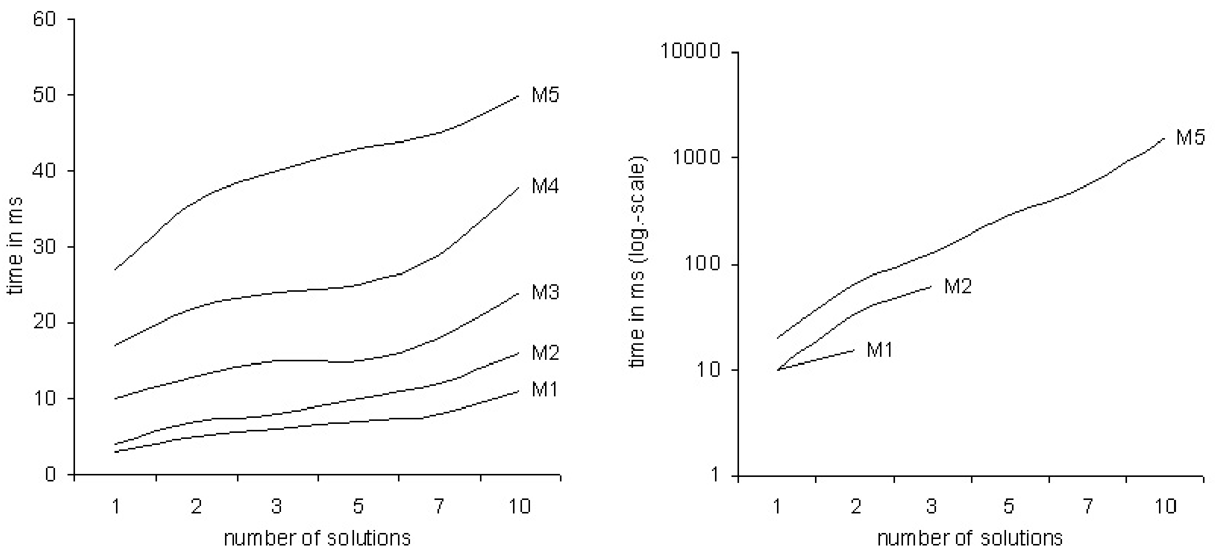


Figure 5. Results for solution strategies *1-different* (left) and *all-different* (right)

[3] Marko Balabanovic and Yoav Shoham, 'Fab: Content-based, collaborative recommendation', *Communications of the ACM*, **40(3)**, 66–72, (1997).

[4] Robin Burke, 'The Wasabi Personal Shopper: A Case-Based Recommender System', in 11th *Conference on Innovative Applications of Artificial Intelligence (IAAI)*, pp. 844–849, Trento, IT, (2000). AAAI.

[5] Robin Burke, 'Hybrid recommender systems: Survey and experiments', *User Modeling and User-Adapted Interaction*, **12(4)**, 331–370, (2002).

[6] Gerhard Fleischanderl, Gerhard Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, **17**(July-August), 59–68, (1998).

[7] Dietmar Jannach, 'Advisor suite - a knowledge-based sales advisory system', in 16th *European Conference on Artificial Intelligence - Prestigious Applications of AI (PAIS)*, ed., L. Saitta Lopez de Mantaras, pp. 720–724. IOS Press, (2004).

[8] Francois Laburthe, Narendra Jussien, Rochart Guillaume, and Cambazard Hadrien, *Choco Tutorial, Sourceforge Open Source*, http://choco.sourceforge.net/tut_base.html.

[9] Greg Linden, Steve Hanks, and Neal Lesh, 'Interactive assessment of user preference models: The automated travel assistant', in 5th *International Conference on User Modeling (UM)*, Lyon, France, (1997).

[10] Daniel Mailharro, 'A classification and constraint-based framework for configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **12**, 383–397, (1998).

[11] Sanjay Mittal and Felix Frayman, 'Toward a generic model of configuration tasks', in 11th *International Joint Conferences on Artificial Intelligence*, pp. 1395–1401, Menlo Park, California, (1989).

[12] David O'Sullivan, Barry Smyth, and David Wilson, 'Understanding case-based recommendation: A similarity knowledge perspective', *International Journal of Artificial Intelligence Tools*, (2005).

[13] M. Pazzani, 'A framework for collaborative, content-based and demographic filtering', *Artificial Intelligence Review*, **13**(5/6), 393–408, (1999).

[14] Pearl Pu and Boi Faltings, 'Decision tradeoff using example-critiquing and constraint programming', *Constraints*, **9**, 289–310, (2004).

[15] P. Resnick, N. Iacovou, N. Suchak, P. Bergstrom, and J. Riedl, 'Grouplens: An open architecture for collaborative filtering of netnews', in *Computer Supported Collaborative Work (CSCW)*, Chapel Hill, NC, (1994).

[16] Francesco Ricci and Hannes Werthner, 'Case base querying for travel planning recommendation', *Information Technology and Tourism*, **3**, 215–266, (2002).

[17] Daniel Sabin and Rainer Weigel, 'Product configuration frameworks - a survey', *IEEE Intelligent Systems*, **17**(July/August), 42–49, (1998).

[18] Hideo Shimazu, 'Expert clerk: Navigating shoppers' buying process with the combination of asking and proposing', in 17th *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1443–1448, (2001).

[19] Barry Smyth, Lorraine McGinty, James Reilly, and Kevin McCarthy, 'Compound critiques for conversational recommender systems', in *IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pp. 145–151, Washington, DC, USA, (2004). IEEE Computer Society.

[20] Edward Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London, UK, 1993.

[21] Paolo Viappiani, Boi Faltings, and Pearl Pu, 'Evaluating preference-based search tools: a tale of two approaches', in 22th *National Conference on Artificial Intelligence(AAAI)*, (2006).

[22] Paolo Viappiani, Boi Faltings, and Pearl Pu, 'Preference-based search using example-critiquing with suggestions', *Artificial Intelligence Research*, **27**, 465–503, (2006).

[23] Markus Zanker and Markus Jessenitschnig, 'Iseller - a generic and hybrid recommendation system for interactive selling scenarios', in 15th *European Conference on Information Systems (ECIS)*, St. Gallen, Switzerland, (2007).

[24] Markus Zanker, Markus Jessenitschnig, Dietmar Jannach, and Sergiu Gordea, 'Comparing recommendation strategies in a commercial context', *IEEE Intelligent Systems*, **22**(May-June), 69–73, (2007).

Towards Recommending Configurable Offerings

Tiihonen Juha [1] and **Felfernig Alexander** [2]

Abstract. Configuration technologies provide a solid basis for the implementation of a Mass Customization strategy. A side-effect of this strategy is that the offering of highly variant products and services triggers the phenomenon of Mass Confusion, i.e., customers are overwhelmed by the size and complexity of the offered assortments. In this context, recommendation technologies can provide help by supporting users in the identification of products and services fitting their wishes and needs. Recommendation technologies have been intensively exploited for the recommendation of simple products such as books or movies but have (with a few exceptions) not been applied to the recommendation of complex products and services such as computers or financial services. In this paper we show how to integrate case-based and content-based recommendation approaches with knowledge-based configurators.

1 Introduction

In many domains, customers do not know and understand in detail the complete set of options supported by a given configurable product. On the one hand configuration options are represented on a rather technical level and customers are overwhelmed by the offered set of alternatives [6] - this phenomenon is well known as Mass Confusion [14]. On the other hand customers do not know their preferences beforehand since preferences are typically constructed [13] within the scope of a configuration session. Even experienced sales persons tend to propose configurations they are used to thus overlooking configuration alternatives which better suit to the customers' wishes and needs. This can cause unsatisfied customers as well as the sales of less profitable configurations. Consequently, users of configuration systems are in the need of more intuitive interaction mechansisms effectively supporting the configuration and selection of interesting product and service alternatives. With a few exceptions [5, 12, 16] existing recommender technologies [8] are primarily applied for the recommendation of simple products and services such as books, movies or compact discs. These technologies have not been integrated into configuration environments dealing with complex products and services. The major goal of this paper is to discuss scenarios in which recommendation technologies can be exploited in configuration sessions. This work has been conducted within the scope of the COSMOS project[3].

[1] Deptartment of Computer Science and Engineering, Helsinki U of Technology

[2] Intelligent Systems and Business Informatics, U Klagenfurt

[3] COSMOS (Customer-Oriented Systematically Managed Service Offerings) is a project supported by the Finnish Funding Agency for Technology and Innovation.

Recommendation Technologies. *Collaborative filtering* (see, e.g., [1]) is one of the most commonly used recommendation technologies. It provides recommendations on the basis of opinions of users (e.g., ratings or purchasing data). A similarity function on opinions about items is exploited to calculate nearest neighbors which are users with similar preference structures. The basic idea is to identify similar users and to recommend their highly rated items that are unknown to the active user. In many B2C scenarios an individual user may purchase too few configurable products to establish a dense enough user profile to be suitable as a basis for pure collaborative filtering. However, collaborative filtering technologies can be used in very specific settings supporting the collaborative development of *product innovations* [10]. This issue will not be further discussed in this paper.

Content based filtering (see, e.g., [15]) approaches recommend items similar to those that the active user has preferred in the past. Items are described by a number of keywords or features. A user model contains previous opinions about items, often presented as keywords or features. A similarity function is used to calculate nearest neighbors [4] which are in this case those items with the highest similarity compared to the given preference information in the user profile. The approach is typically applied for recommending text-based items such as articles or web pages. One challenge of applying this technique to configurable offerings is that - beside the availability of textual component descriptions - building a user profile requires repetitive configurations of one user. Furthermore, a profile may soon become outdated in rapidly evolving domains such as PC's. Beside the calculation of the k-nearest neighbors predictions [4] on interesting items, recommendations can be based on naive Bayes predictors [15] which allow the prediction of interesting items on the basis of their probability of being selected given the existing user preferences. *In this paper we will focus on the application of naive Bayes predictors for the identification of interesting configurations.*

Utility based recommendation estimates the utility of an item for a customer and recommends items with the highest utility. Domain-specific interest dimensions have to be identified in this context. For PC's, interest dimensions could be *economy*, *reliability*, *graphics performance* and *weight.* Items are given numeric utility values with respect to the interest dimensions. The user specifies his preferences in terms of importance (weight) of each interest dimension. Given this information, item utilities can be computed for the active user. An example for the application of utility-based recommendation approaches in the financial services domain is given in [9]. A further discussion of utility-based approaches in the configuration context is outside the scope of this paper.

Knowledge-based recommenders exploit explicit information about items and explicit knowledge on how user requirements on those items can be specified [4][9]. *Constraint-based recommendation* is a knowledge-based approach where alternative items and potential customer requirements are described on the basis of a set of features and the corresponding constraints. Filter constraints match customer requirements to suitable items. Compatibility constraints ensure the consistency of requirements. To resolve inconsistencies, explanation and repair functionalities are provided [9]. Constraint-based approaches exploit the same technologies as many knowledge-based configuration environments and are additionally combined with, e.g., utility-based recommendation approaches supporting the ranking of candidate configurations. *Case-based recommendation* [3] is another type of knowledge-based recommendation. It exploits similarity functions to determine items fitting to the wishes an needs of users. In contrast to content-based filtering and collaborative filtering those similarity functions compare elementary properties of items (e.g., PC price) rather than extracted keywords or categories. Our major goal in this paper is to apply and extend different concepts of case-based and content-based recommendation in order to make them applicable in configuration settings.

Recommendation Scenarios. Recommendation of configurable offerings can focus on

- selecting a suitable base product line to configure (such as a car model)
- recommending a complete configuration (such as a complete PC for gaming or a tractor for peat harvesting including suitable wheels, air-intake filters, and other equipment)
- recommending how to complete a configuration (e.g., to propose still unspecified details of a PC)
- recommending a subconfiguration (e.g., a storage subsystem suitable for a particular type of use such as a PC storage subsystem for full-HD video-editing and authoring)
- recommending individual attribute or component settings (e.g., a mobile data connection for a business person)

Consequently, a high diversity of usage and integration scenarios for recommendation technologies in the configuration context can be envisioned. In this paper our major focus is the integration of case-based and content-based recommendation into existing configurators. We analyze existing approaches in the field [5, 12, 16] and show how to extend and improve those approaches by developing a more sophisticated notion of *similarity between item features* and *explicitly taking into account customer importance weights* for features.

The remainder of the paper is organized as follows. In the following section we introduce a working example from the domain of configurable computers. In Section 3 we discuss relevant case-based and content-based algorithms for configurable products and services and introduce our extensions to those algorithms. A discussion of related and future work is presented in Section 4. Section 5 concludes the paper.

2 Working Example

In this paper we consider (for reasons of simplicity) only "flat" configuration models consisting of features (no variation of structure or connections), each having a finite domain of possible values. Our example product is a PC, that has as features a motherboard (*mb*), a hard disk (*hd*), an optical drive (*od*), a processor (*pr*), and optionally a graphics card (*gc*). The amount of memory (*me*) is specified in gigabytes (1, 2, 3, or 4). A *complete configuration* specifies a value for each feature. Furthermore, a *valid configuration* is complete and consistent with a defined set of constraints.

We represent some non-configurable attribute values related to features to specify constraints more intuitively. Processors are introduced in Table 1(a). Processor performance is approximated with an industry standard benchmark, specified by *CScr*. *Socket* determines a processor's connection to a motherboard. Motherboards (see Table 1(b)) are designed to be compatible with either manufacturer A's or I's processors, socket 'a' or 'i', respectively. Thus, a specific constraint specifies that a processor must fit the motherboard: $pr.socket = mb.socket$. In addition, some motherboards provide an integrated graphics card ($IntGr = yes$).

pr	socket	CScr
as	a	1250
i4	i	2858
i9	i	4537

mb	socket	IntGr	GScr
a1	a	yes	300
a2	a	no	0
i1	i	yes	200
i2	i	no	0

Table 1. Processors (a, left) and motherboards (b, right)

Separate graphics cards provide higher performance than those integrated to motherboards. Graphics performance is approximated with an industry standard benchmark, represented by graphics performance score (*GScr*). Similarly, motherboards with integrated graphics card specify their graphics performance with *GScr*. Graphics cards *g*2, *g*8, and *g*9 have graphics performance scores 2800, 2200, and 5500, respectively. A system must always have a way to produce graphics. Thus the constraint $(mb.IntGr = no) \implies (gc \neq no)$ is introduced. pc.GScr refers to graphics performance of the PC, determined as the maximum *GScr* provided by the graphics card or the motherboard.

Hard disks (*hd*) are available in different capacities (GB) ($h2.capacity = 250$, $h5.capacity = 500$, and $h9.capacity = 1000$). All optical drives (see Table 2) read CD and DVD. Some write DVD or DVD + Blu-ray.

od	Write DVD & CD (dw)	Read Blu-ray (br)	Write Blu-ray (bw)
dr	no	no	no
dw	yes	no	no
br	no	yes	no
bw	yes	yes	yes

Table 2. Properties of optical drives of the running example

Three additional features, namely video editing (*vi*), photos (*ph*), and gaming (*ga*) are included in the configuration model to describe intended use of the PC being configured. Details are specified in Table 3.

The following domain knowledge is available:
$ph \neq no \implies od.dw = yes$: to archive photos
$ph = adv \implies hd.capacity \geq 500$: disk space for photos
$ph = adv \implies pr.CScr \geq 2500$: CPU for advanced photo

Feature	Values		
Video editing (**vi**)	no (*no*)	standard definition (*sd*)	high-definition (*hd*)
Photos (**ph**)	no (*no*)	normal home use (*std*)	advanced amateur or professional (*adv*)
Gaming (**ga**)	no or 2D games (*2d*)	3d games (*3d*)	enthusiast performance 3d games, HD resolutions (*adv*)

Table 3. Intended usage features of the running example

$ph = adv \implies me \geq 2$: RAM for advanced image processing
$vi = sd \implies pr.CScr \geq 2700$: CPU for SD video editing
$vi = hd \implies pr.CScr \geq 4500$: CPU for HD video editing
$vi = sd \implies od.dw = yes$: burn DVD videos
$vi = hd \implies od.bw = yes$: burn Blu-ray videos
$ga = 3d \implies pr.CScr \geq 1500$: CPU for 3D gaming
$ga = 3d \implies pc.GScr \geq 1500$: graphics for 3D gaming
$ga = adv \implies pr.CScr \geq 2800$: CPU for advanced gaming
$ga = adv \implies pc.GScr \geq 5000$: graphics for advanced gaming

For the formulas discussed in this paper, we assume the following distribution of *feature importance weights*: video editing $w(vi) = 5\%$, photos $w(ph) = 5\%$, gaming $w(ga) = 9\%$, processor $w(pr) = 18\%$, motherboard $w(mb) = 5\%$, amount of memory $w(me) = 15\%$, hard disk $w(hd) = 16\%$, graphics card $w(gc) = 17\%$, optical drive $w(od)$=10%. Those features could stem from direct customer specifications, representative preferences from statistical samples, or the application of utility constraints as documented in [9].

Notation. We base the discussion of recommendation algorithms on the following conventions. Relation $Conf$ holds previous K configurations, each having values for the existing N features $f_1, .., f_N$. The value of feature f_i in configuration k is referred to as $f_{i,k}$. The k^{th}configuration is referred to as $Conf_k$. Classification (discussed in Section 3.1) of configuration k is referred to as c_k. When referring to the profile of the active user, we use index u, $u \notin K$, e.g., $f_{i,u}$ refers to the value of feature i for the active user. The set of specified features in the active user profile is F_u. $F_u = \{f_j | f_{j,u} \neq noval\}$, and the set of features for which the active user profile does not have a value is $\bar{F}_u$. Furthermore, a projection $\pi_{F_u}(Conf)$ of previous configurations for which the profile has (does not have) values is referred to as $Conf_{F_u}$ ($Conf_{\bar{F}_u}$). The index of a configuration (k) is considered to be included in $Conf_k$ and projections, see Table 4, to avoid unintended removal of duplicate tuples. Finally $dom(f_j)$ returns the domain of f_j.

k	f_1 **vi**	f_2 **ph**	f_3 **ga**	f_4 **pr**	f_5 **mb**	f_6 **me**	f_7 **hd**	f_8 **gc**	f_9 **od**	c
1	no	no	2d	as	a1	1	h2	no	dr	ba
2	no	std	2d	as	a2	1	h5	g2	dw	st
3	sd	std	adv	i4	i2	3	h5	g9	dw	ad
4	hd	adv	adv	i9	i2	4	h9	g9	bw	ad
5	sd	adv	3d	i4	i1	2	h9	g8	dw	st
u	no	no	3d							

Table 4. Previous configurations and active user profile

3 Recommendation Algorithms

3.1 Distance Metrics

The algorithms discussed in this paper use distance functions to determine similarity or dissimilarity of individual feature values, and ultimately that of configurations. The motivation for using distance functions instead of equality when comparing feature values is that equality may be too strict for a measure - close values or configurations could remain ignored.

We decided to include ideas of the *Heterogeneous Value Difference Metric* ($HVDM$) [17] to cope with symbolic (nominal) and numeric features in a relatively simple manner. On the feature level the distance is defined as follows where the function $d_{f_i}(x, y)$ returns the distance between values x and y of feature a, using a different sub-function for different types of features: distances between symbolic feature values are computed by function $vdm_{f_i}(x, y)$, and those between linear values by function $diff_{f_i}(x, y)$. Based on experiments of [17], these functions provide similar influence on the overall distance measurements. Distance values returned by $d_{f_i}(x, y)$ are *normalized* to usually (but not always) be in range 0 to 1.

$$d_{f_i}(x, y) = \begin{cases} 1 \quad \text{if x or y is unknown; otherwise} \\ vdm_{f_i}(x, y), \quad \text{if } f_i \text{ is symbolic} \\ diff_{f_i}(x, y), \quad \text{if } f_i \text{ is linear} \end{cases}$$

The function $vdm_{f_i}(x, y)$ learns the similarity of symbolic values in a domain automatically. This takes place by examining the probability that individual feature values contribute to the classification of the sample vector - in our case classification of configurations. Slightly oversimplifying, the closer the probability of a pair of feature values to be present in identically classified configurations, the more similar these feature values are considered. In this paper we take a simplistic view, and consider a configuration to belong to one of three clusters (basic (ba), standard (st), or advanced (ad)), which is used as the classifier for $HVDM$ (see column "c" in Table 4).

$$\begin{aligned} vdm_{f_i}(x, y) &= \sqrt{\sum_{c=1}^{C} \left| \frac{N_{f_i,x,c}}{N_{f_i,x}} - \frac{N_{f_i,y,c}}{N_{f_i,y}} \right|^2} \\ &= \sqrt{\sum_{c=1}^{C} |P_{f_i,x,c} - P_{f_i,y,c}|^2} \end{aligned}$$

In the function $vdm_{f_i}(x,y)$, $N_{f_i,x}$ is the number of instances (configurations) in the training set T that have value x for feature f_i; $N_{f_i,x,c}$ is the number of instances in T that have value x for feature f_i and output class c; C is the number of output classes in the problem domain (in our case 3 - see Table 4). $P_{f_i,x,c}$ is the conditional probability of output class c given that feature f_i has the value x, i.e., $P(c|f_i = x)$. When $N_{f_i,x} = 0$, $P(c|f_i = x)$ is considered 0.

For example, in our example population $Conf$, $N_{pr,as} = 2$, $N_{pr,i4} = 2$, and $N_{pr,i9} = 1$. The classification frequencies for processor $N_{pr,x,c}$ are presented on left half of Table 5, e.g., feature value as for classification basic (ba) occurs exactly once. The resulting distance matrix for processors is presented on the right half of Table 5.

Distances between linear values x and y of feature f_i (in $Conf$) are determined by function $diff_{f_i}(x, y)$. Since 95% of the values in a normal distribution fall within two standard deviations of the mean, the difference between numeric values

c,x	as	i4	i9
ba	1	0	0
std	1	1	0
adv	0	1	1

	as	i4	i9
as	0	0.707	1.225
i4	0.707	0	0.707
i9	1.225	0.707	0

Table 5. Classification frequencies of pr (left), and corresponding distance matrix (right)

is divided by 4 standard deviations to scale each value into a range that is usually (95% of cases) of width 1. As motivated in [17] "distances are often normalized by dividing the distance for each variable by the range of that attribute, so that the distance for each input variable is in the range 0..1. However, dividing by the range allows outliers (extreme values) to have a profound effect on the contribution of an attribute. For example, if a variable has values which are in the range 0..10 in almost every case but with one exceptional (and possibly erroneous) value of 50, then dividing by the range would almost always result in a value less than 0.2."

$$diff_{f_i}(x, y) = \frac{|x - y|}{4\sigma_{f_i}}$$

The distance metric $d_{f_i}(x, y)$ usually returns a value from 0 to 1. The similarity $sim_{f_i}(x, y)$ between two feature values x and y of feature f_i can then be defined as $sim_{f_i}(x, y) = 1 - d_{f_i}(x, y)$. In the following subsections we show how these metrics can be applied to the recommendation of feature values as well as to the recommendation of complete configurations. In the sense of case-based configuration we investigate existing (and similar) configurations in order to predict interesting feature settings and complete solutions. Note that our cases describe user preferences as well as technical features.

Following this approach, we extend existing algorithms [5] to take into account *similarity* of feature values, not just direct equality. Furthermore, we extend previous approaches to take into account the *weights* of different features.

3.2 Nearest Neighbor

The idea of a *nearest neighbor* is simple: determine a neighbor configuration, which is closest to the known parts of active user's profile, and recommend feature values of this nearest neighbor. The nearest neighbor is determined as follows (distance between two configurations):

$$dist(Conf_u, Conf_a) = \sum_{i \in F_u} d_{f_i}(f_{i,u}, f_{i,a}) * w(f_i)$$

For nearest neighbor c the following has to hold: $\nexists c', 1 \leq c' \leq K : dist(Conf_u, Conf_{c'}) < dist(Conf_u, Conf_c)$.

In our example, the nearest neighbor relative to our user profile is $Conf_1$: $d_{vi}(no, no)$= 0.000, $w(vi)$=0.050; $d_{ph}(no, no)$ = 0.000, $w(ph)$=0.050; $d_{ga}(3d, 2d)$ = 0.707, $w(ga)$=0.090. Total weighted distance $dist(Conf_u, Conf_1) = 0.064$. Applying the nearest neighbor formula to configurations $Conf_2$ .. $Conf_5$ provides distances 0.125, 0.224, 0.250, and 0.097. Unfortunately, the combination of known feature values of the user profile and $Conf_1$ is not consistent (graphics and cpu performances are not sufficient for 3d gaming). Feature values of the nearest consistent neighbor $Conf_5$ are recommended: $pr = i4$, $mb = i1$, $me = 2$, $hd = h9$, $gc = g8$, $od = dw$.

3.3 Weighted Majority Voter

The *weighted majority voter* [5] recommends individual feature values based on each neighbor configuration in $Conf$ "voting" for its feature values. The weight of each neighbor vote is determined by the number of equal feature values to the user profile. For example, the weight of $Conf_1$ for user u is 2, because $f_{vi,1} = f_{vi,u} = no$, and $f_{ph,1} = f_{ph,u} = no$. Thus, $Conf_1$ would give 2 votes for $pr = as$, $mb = a1$, $me = 1$, $hd = h2$, $gc = no$, and $od = dr$. The following feature values get most votes: $pr = as$ (3 votes), $mb = a1$ (2), $me = 1$ (3), $hd = h2$ (2), $gc = no$ (2), and $od = dr$ (2).

The consistency of a potential recommendation is checked by adding the proposed value to the known values in the user profile. After user selects a feature value, recommendations will be recalculated to reflect the new situation. Due to consistency checks, $pr = i4$ (1 vote) and $gc = g2$ (1 vote) replace those with most votes, assuming the selection of the first feature value in a domain in case of a tie in votes.

Next, we rewrite the formula of [5] and propose the corresponding extensions. First, we define the equality function eq to return 1 when two values are equal, otherwise 0.

$$eq(x, y) = \begin{cases} 1 & \text{if x=y} \\ 0 & \text{otherwise} \end{cases}$$

The weight $w(conf_x, conf_u)$ of a neighbor configuration $conf_x$ with respect to configuration $conf_u$ (a user's partial configuration) is the number of equal feature values in features for which $conf_u$ has a value (F_u):

$$w(conf_x, conf_u) = \sum_{i \in F_u} eq(f_{i,x}, f_{i,u})$$

The prediction score $pr(conf_u, f_j, v)$ for (user's configuration) $conf_u$ having value v for feature f_j is thus sum of weights (votes) of neighbors having value v for feature f_j:

$$pr(conf_u, f_j, v) = \sum_{i=1}^{K} eq(f_{j,i}, v) * w(conf_i, conf_u)$$

A value v with maximum prediction score $pr(conf_u, f_j, v)$ is the recommendation $r(f_j, conf_u)$ for feature f_j in configuration $conf_u$:

$$r(f_j, conf_u) = v \text{ such that}$$
$$\nexists v', v' \in dom(f_j) : pr(conf_u, f_j, v') > pr(conf_u, f_j, v)$$

We propose a modified algorithm to derive neighbor weights. Neighbor weights are determined by the similarity of neighbor and user profile feature values instead of equality used by [5]. Thus, similar values compared to user's existing selections contribute to the weight of a neighbor. Further, we take into account the importance of individual features for a user (feature weights). Both of those aspects are extremely important for interactive settings. Thus, the weight $w(conf_x, conf_u)$ of a neighbor configuration $conf_x$ with respect to configuration $conf_u$ is defined as follows:

$$w(conf_x, conf_u) = \sum_{i \in F_u} sim_{f_i}(f_{i,x}, f_{i,u}) * w(f_i)$$

The weights of example neighbors are $w(conf_1, conf_u) = 0.126$, $w(conf_2, conf_u) = 0,065$, $w(conf_3, conf_u) = -0.034$, $w(conf_4, conf_u) = -0.060$, and $w(conf_5, conf_u) = 0.093$. Using these weights, the first potential recommendations are $pr = as$ (0.191), $mb = a1$ (0.126), $me = 1$ (0.191), $hd = h2$ (0.126), $gc = no$ (0.126), and $od = dr$ (0.126). To provide (locally) consistent recommendations, pr and gc must be substituted with $pr = i4$ (0.060), and $gc = g8$ (0.093).

3.4 Most Popular Choice

The most popular choice algorithm [5] recommends entire remaining configurations, typically to complete a configuration. The probability estimate for a configuration $c \in Conf$ with respect to known features in user's profile F_u (and corresponding unknown features $\bar{F_u}$) is calculated as:

$$Pr(c, u, F_u) = Pr_{basic}(c, \bar{F_u}) * \prod_{j \in F_u} Pr(f_{j,u} = f_{j,u}|Conf)$$

In the original formula, basic probability $Pr_{basic}(c, \bar{F_u})$ of a neighbor configuration c is based on the popularity of it's feature values on features for which the user profile does not have a value, $\bar{F_u}$. The basic probability for feature f_j having the value that configuration c has $(f_{j,c})$ is simply the proportion of neighbors having that value for feature f_j. Basic probability of a configuration is determined by multiplying the basic probabilities for its feature values. We apply function $count(f_j, v)$ that returns the number of neighbors in $Conf$ having value v for feature f_j: $count(f_j, v) = \sum_{k=1}^{K} eq(f_{j,k}, v)$.

$$Pr_{basic}(c, \bar{F_u}) = \prod_{j \in \bar{F_u}} \frac{count(f_j, f_{j,c})}{K}$$

We extend the concept of basic probability by giving each feature value support when neighbour configurations have feature values within maximum distance Δ, $(d_{f_j}(f_{j,u}, f_{j,a}) \leq \Delta)$. The support is defined as term $(1 - d_{f_j}(f_{j,u}, f_{j,a}))^2$ to quickly lessen its significance when the distance increases. We define support $s_{f_j}(x, y)$

$$s_{f_j}(x, y) = \begin{cases} (1 - d_{f_j}(x, y))^2, & \text{if } d_{f_j}(x, y) \leq \Delta \\ 0, & \text{otherwise} \end{cases}$$

Maintaining these as probabilities requires that the sum of probabilities of existing values is 1. Thus, the sum of supports for feature f_j having the value that configuration c has $(f_{j,c})$ is divided by the sum of supports given to all values in domain of f_j that exist in at least one neighbor configuration. Therefore

$$Pr_{basic}(c, \bar{F_u}) = \prod_{j \in \bar{F_u}} \frac{\sum_{k=1}^{K} s_{f_j}(f_{j,c}, f_{j,k})}{\sum_{v \in dom(f_j)} \sum_{k=1}^{K} s_{f_j}(v, f_{j,k}) * min(1, count(f_j, v))}$$

The basic probability is weighted with a Bayesian predictor for the user profile u to have the values already selected, given the existing neighbors, $\prod_{j \in F_u} P(f_{j,u} = f_{j,u}|Conf)$. $P(f_{j,u} = f_{j,u}|Conf)$, is defined to be an m-estimate [2] to stabilize probability calculations even in case of (too) few samples. The m-estimate assumes m virtual samples with initial probability p that augment the estimation of probability. $m_{est}(N_c, N, p, m) = \frac{N_c + mp}{N + m}$. We apply the following m-estimate parameters: 1) the number of "in-samples" N_c is the number of such neighbor configurations that have equal value (f_j, u) as the user profile for feature f_j being inspected, and that have the same configuration with respect to features $(\bar{F_u})$ that the user profile does not have a value; 2) the number of all samples N is the number of such neighbor configurations that have the same configuration with respect to those features $(\bar{F_u})$ that the user profile does not have a value; and 3) m-estimate virtual sample parameters are $p = 1/K$, and $m = K$.

$$\prod_{f_j \in F} P(f_{j,u} = f_{j,u}|Conf) =$$

$$\prod_{f_j \in F} m_{est}(eqcfgs_m(c, F \cup f_j, f_j, f_{j,u}), eqcfgs(c, F), 1/K, K)$$

$eqcfg(i, j, F)$ tests if neighbor configurations i and j are equal with respect to a set of features F. It returns 1 iff profiles i and j have equal feature values for all features $f \in F$. Otherwise it returns 0.

$$eqcfg(i, j, F) = \begin{cases} 1 & \text{if } \forall f \in F : eq(f_{f,i}, f_{f,j}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$eqcfgs(c, F)$ returns the number of neighbor configurations that are equal to configuration c with respect to a set of features F. $eqcfgs(c, F) = \sum_{k=1}^{K} eqcfg(c, k, F)$.

$eqcfgs_m(c, F, f_j, v)$ returns the number of neighbor configurations that are equal to configuration c with respect to a set of features F, and which have value v for feature f_j. $eqcfgs_m(c, F, f_j, v) = \sum_{k=1}^{K} eqcfg(c, k, F) * eq(f_{j,k}, v)$.

In our example, the basic probability of $conf_5$ with the original formula [5] is $0.003072 = 0.4 * 0.2 * 0.2 * 0.4 * 0.2 * 0.6$, because $f_{pr} = i4$ two times, other terms for basic probability are 0.2 (mb), 0.2 (me), 0.4 (hd), 0.2 (gc), and 0.6 (od). With our modified formula and (a large) $\Delta = 0.8$ they are: 0.403 (pr), 0.286 (mb), 0.280(me), 0.444 (hd), 0.286 (gc) 0,600 (od) $= 0{,}00246$

$\prod_{j \in F_u} P(f_{j,u} = f_{j,u}|Conf) = m_{est}(1, 1, 0.2, 5) * m_{est}(0, 1, 0.2, 5) * m_{est}(0, 1, 0.2, 5)$. For example, for the first term $N_c = 1$, because configuration $conf_5$ has $vi = sd$ just as the user profile, and there are no other equal configurations to $conf_5$ with respect to features absent from user profile. The second term for ph has $N_c = 0$, because $conf_5$ has different value for ph than the user profile u. $conf_5$ becomes the configuration of choice (with estimate 1.12E-07, thus its value are recommended: $pr = i4$, $mb = i1$, $me = 2$, $hd = h9$, $gc = g8$, and $od = dw$.

4 Related and Future Work

The paper of [5] presents a recommender implementation for on-line PC configuration and underlying recommendation algorithms. We extended the recommendation algorithms of [5] in order to be able to take into account the aspects of importance weights and similarity (which substitutes the notion of equality). Furthermore, we take into account the notion of consistency, i.e., we only allow recommendations which are consistent with the given set of customer requirements. Admittedly, the evaluation of our approach has not been completed up to now but will be a strong focus of future work.

The contribution of [12] presents an approach to integrate case-based reasoning with constraint solving with the goal to adapt identified nearest neighbors to the new configuration problem. The used algorithm for calculating nearest neighbors takes into account component structures but does not take into account probabilities of selection. The authors then discuss an approach to the calculation of adaptations for the identified nearest neighbors in order to conform with the new customer preferences. No details are provided regarding the minimality of changes or how the adaption effects the given customer requirements. One of our major goals for future research is to integrate mechanisms which allow the calculation of minimal adaptations in the case of recommendations partially inconsistent with the given customer requirements.

[16] present an approach to the application of case-based reasoning for product configuration tasks. The authors develop their approach on the basis of a high-level product structure where instance similarities are determined on the basis of simple equality relations. Compared to the approach presented in this paper, the authors do not take into account propabilities which provide an indication of the most promising similar cases.

Management of consistency of recommendations with respect to an existing (partial) configuration is an important topic. In cases where there exist interesting configurations for customers but those are incompatible with the initial set of requirements, corresponding explanations have to be provided. Vice versa, such explanations should take into account minimal distances to existing nearest neighbours. The developments in this area will rely on existing work related to the determination of explanations [11, 7].

We proposed to apply a similarity metric that automatically determines the similarity of feature values based on classification outcomes of configurations. This approach has potential to improve the quality of identified cases and feature settings. However, efforts have to be invested to evaluate the proposed metrics within the scope of user studies. An interesting open question in this context is whether classifiers should be based on the whole configurable product, or should, e.g., sub-system-specific classifiers be applied to provide more accurate similarity metrics.

Reconfiguration of products and services could benefit from personalized recommendation support. For example, in insurance and financial domains situation or needs of individuals or customer organizations change within long relationships of customership. It should be possible to update the configured solution correspondingly while avoiding solutions that introduce sub-optimal switching costs or weakening of current contractual terms. These are open questions which are within the focus of the COSMOS project.

5 Conclusions

In this paper we have shown the potential benefitss of integrating case-based/content-based recommendation with configuration technologies. This integration allows for the derivation of individualized and personalized product and service offerings. Those technologies show great potential for reducing the so-called mass confusion phenomenon which prevents users from identifying products and services fitting their wishes and needs. The recommendation approach presented in this paper is a first but important step towards personalized configuration systems which more actively support users in preference construction processes.

References

[1] G. Adomavicius and A. Tuzhilin, 'Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions', *Knowledge and Data Engineering, IEEE Transactions on*, **17**(6), 734–749, (2005).

[2] I. Bratko, B. Cestnik, and I. Kononenko, 'Attribute-based learning', *AI Communications*, **9**(1), 27–32, (1996).

[3] R. Burke, 'Knowledge-based Recommender Systems', *Encyclopedia of Library and Information Systems*, **69**(32), (2000).

[4] R. Burke, 'Hybrid Recommender Systems: Survey and Experiments', *User Modeling and User-Adapted Interaction*, **12**(4), 331–370, (2002).

[5] R. Coester, A. Gustavsson, R. Olsson, and A. Rudstroem, 'Enhancing web-based configuration with recommendations and cluster-based help', in *AH'02 Worksh. on Recommendation and Personalization in EComm.*, Malaga, Spain, (2002).

[6] W. Emde, C. Beilken, J. Boerding, W. Orth, U. Ptersen, J. Rahmer, M. SPenke, A. Voss, and S. Wrobel, 'Configuration of Telecommunication Systems in KIKon', in *Workshop on Configuration*, pp. 105–110, Stanford, California, (1996).

[7] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner, 'Consistency-based Diagnosis of Configuration Knowledge Bases', *Artificial Intelligence*, **2**(152), 213–234, (2004).

[8] A. Felfernig, G. Friedrich, and L. Schmidt-Thieme, 'Recommender systems', *IEEE Intelligent Systems-Special Issue on Recommender Systems*, **22**(3), (2007).

[9] A. Felfernig, K. Isak, K. Szabo, and P. Zachar, 'The vita financial services sales support environment', in *AAAI*, pp. 1692–1699, (2007).

[10] N. Franke, P. Keinz, and M. Schreier, 'Complementing mass customization toolkits with user communities', *Journal of Product Innovation Management*, forthcoming, (2008).

[11] G. Friedrich, 'Elimination of Spurious Explanations', in *16th European Conference on Artificial Intelligence (ECAI 2004)*, eds., G. Müller and K. Lin, pp. 813–817, Valencia, Spain, (2004).

[12] L. Geneste and M. Ruet, 'Experience based configuration', in *17th International Conference on Artificial Intelligence*, volume 1, pp. 4–10. IJCAI, (2001).

[13] G. Haeubl and K.B. Murray, 'Preference Construction and Persistence in Digital Marketplaces: The Role of Electronic Recommendation Agents', *Journal of Consumer Psychology*, **13**(1), 75–91, (2003).

[14] C. Huffman and B. E. Kahn, 'Variety for sale: Mass customization or mass confusion?', *Journal of Retailing*, **74**(4), 491–513, (1998).

[15] M.J. Pazzani and D. Billsus, 'Content-based recommendation systems', *The Adaptive Web: Methods and Strategies of Web Personalization, Lecture Notes in Computer Science*, **4321**, (2006).

[16] H. Tseng, C. Chang, and S. Chang, 'Applying case-based reasoning for product configuration in mass customization environments', *Expert Sys. with Applic.*, **29**(4), 913–925, (2005).

[17] D. Wilson and T. Martinez, 'Improved Heterogeneous Distance Functions', *Journal of Artificial Intelligence Research*, **6**, 1–34, (1997).

Beyond Valid Domains in Interactive Configuration

Tarik Hadzic and Barry O'Sullivan[1]

Abstract. A key requirement in interactive configuration is the capability of the configurator to prevent the user from selecting options that do not lead to valid configurations. Each time the user selects a value for an attribute of a configurable product, the configurator should respond by computing and displaying only the valid domains for each unassigned attribute. In this paper we argue that richer forms of user-configurator interaction can take place. Showing only valid domains is unnecessarily restrictive if it is possible to compactly view the solution space over several variables simultaneously. This is particularly the case when the entire solution space is represented by a tractable data-structure, such as multi-valued decision diagrams, when multi-dimensional projections can be efficiently computed. We present an approach to compactly presenting the space of solutions to a configuration problem using a set of Cartesian products of valid domains. We show that even when a complete representation is too large, by using techniques from this paper it is possible to adequately represent the solution space with high compression and small sacrifices in soundness.

1 Introduction

Presenting valid domains to a user at each step in an interactive configuration setting is often presented as the key criterion for a successful product configurator. In this setting, each time the user assigns a value to a product attribute, the configurator responds by calculating and displaying a set of values for each of the unassigned variables that are consistent with at least one solution. By selecting values from such *valid domains*, a user is guaranteed *backtrack-freeness* (every selectable value is part of at least one solution) and *completeness* (every solution is reachable by selecting valid values only).

However, a challenge with this approach is that computing the set of valid domains is often NP-hard, particularly when the configuration problem is represented as a set of constraints. To mitigate this complexity, in order to guarantee efficient online responses to user interactions, knowledge about the product to be configured is often *compiled* in an off-line phase, prior to user interaction, into a tractable representation such as an automaton [AFM02], binary decision diagram [Bry86] or multi-valued decision diagram [HA06]. User interaction has guaranteed worst-case response times in the size of compiled representation.

In this paper we argue that compiled representations allow for much richer forms of decision support than those currently utilized. We argue that we should move away from valid domains computation towards richer forms of solution space visualization and interaction. In particular, showing valid domains is an unnecessarily narrow form of communication between a user and a configurator, if it is possible to compactly view the solution space over several variables simultaneously. This is especially the case when the entire solution space is compiled into a tractable data-structure, such as *multi-valued decision diagrams* (MDDs), where multi-dimensional projections can be efficiently computed. In other words, if we already have information and it can be displayed conveniently, why hide it behind valid domains?

In this paper we present an approach to compactly visualizing the space of solutions to a configuration problem using a set of Cartesian products of valid domains when the solution space is represented as an MDD. We evaluate the approach on a real-world configuration instance, and show that even when a complete display of the entire solution space is too large to show to the user, by applying our techniques it is possible to achieve adequate compression of the data for small sacrifices in soundness.

Our Cartesian products representation is related to [HF92], where a cross product representation of the constraint satisfaction problem was utilized during search for a single or all solutions. However, we use Cartesian products as a visualization device rather than a reasoning mechanism.

In Section 2 we present the background work. In Section 3 we illustrate the main idea of moving beyond the valid domains to using a set of Cartesian products as a richer interaction device with a user. In Section 4 we identify algorithmic challenges associated with the main idea. In Section 5 we present an MDD-based approach for data compression for the purpose of visual interaction. In Section 6 we evaluate our MDD-based techniques on a real-world configuration instance. Finally, in Section 7 we conclude and outline future work.

2 Preliminaries

An interactive configurator assists a user to define a valid configuration by providing feedback on valid options for product attributes. Knowledge about the product or service to be configured can be conveniently represented as a *constraint satisfaction problem* (X, D, C) where configuration constraints $C = \{c_1, \ldots, c_m\}$ are posted over variables $X = \{x_1, \ldots, x_n\}$ with domains $D = \{D_1, \ldots, D_n\}$. A solution to the CSP problem corresponds to a valid configuration.

In an interaction step where a subset of the variables $X' \subseteq X$ has been assigned, we denote with ρ the current user assignment $\rho = \{(x_i, v_i) \mid x_i \in X', v_i \in D_i\}$. The configurator calculates and displays a *valid domain* $VD_i^\rho \subseteq D_i$ for each unassigned variable $x_i \in X \setminus X'$. A domain is *valid* if it contains those and only those values with which ρ can be extended to a total valid assignment. We refer to this configurator feedback as a *calculating valid domains* (CVD) functionality. It delivers important interaction requirements: *backtrack-freeness* (user should never be forced to backtrack) and *completeness* (all valid configurations should be reachable) [HSJ+04]. Computing valid domains in fact is equivalent to

[1] Cork Constraint Computation Centre, University College Cork, Ireland. Email: {t.hadzic,b.osullivan}@4c.ucc.ie

enforcing generalized arc consistency with respect to conjunction of all constraints C.

Since calculating valid domains is an NP-hard problem, it is not possible to guarantee interactive response in *real-time*. Therefore, a popular approach is to *compile* all CSP solutions *off-line* (prior to user interaction) into a tractable datastructure, such as an automaton or a decision diagram, and efficiently compute valid domains using the compiled representation. In this paper we consider compilation into *multi-valued decision diagrams*.

Definition 1 (Multi-Valued Decision Diagram) *A multi-valued decision diagram (MDD) M is a tuple (V, r, E, var), where V is a set of vertices containing the special terminal vertex* **1** *and a root $r \in V$, $E \subseteq V \times V$ is a set of edges such that (V, E) forms a directed acyclic graph with r as the source and 1 as the sink for all maximal paths in the graph. Further, $var : V \rightarrow \{1, \ldots, n+1\}$ is a labeling of all nodes with a variable index such that $var(\mathbf{1}) = n+1$. Each edge $e \in E$ is denoted with a triple (u, u', v) of its start node u, its end node u' and an associated value v.*

We work only with *ordered* MDDs. A total ordering $<$ of the variables is assumed and all edges (u, u', v) respect the ordering, i.e. $var(u) < var(u')$. For convenience we assume that the variables in X are ordered according to their indices. Ordered MDDs can be considered as being arranged in n *layers* of vertices, each layer being labeled with the same variable index. While MDDs in general allow edges skipping variable layers in this paper we consider only MDDs without long edges, i.e. where for each $(u, u', v) \in E$, $var(u') = var(u) + 1$. It is also a matter of taste whether to explicitly represent assignments leading to infeasibility by redirecting them to an additional terminal node **0**, or to have only one terminal **1** and not to represent edges for violating assignments.

Every path between root r and terminal **1** corresponds to a unique solution. We will also consider *meta-paths* between r and **1**, denoted as $p = (u_1, \ldots, u_{n+1})$ where $u_1 = r$ and $u_{n+1} = \mathbf{1}$. Between any two vertices in a meta-path u_i, u_{i+1}, we consider all edges between them, i.e. we consider a set of values $D_{u_i,u_{i+1}} = \{v \mid (u_i, u_{i+1}, v) \in E\}$. A meta-path therefore corresponds to a subset of solutions:

$$Sol(p) = \prod_{1}^{n} D_{u_i,u_{i+1}}.$$

We will in general use Sol to denote the set of solutions represented by an MDD.

3 Compactly Representing Solutions

Showing valid domains for each variable is a narrow way of interacting with the user. In order to completely specify a solution, a user is forced to consider one variable at a time even when it is possible to compactly view the solution space over several variables simultaneously. The standard *calculation of valid domains* is a display of one-dimensional projections Sol_{x_i} for each variable x_i. We argue that there is no reason to "hide" the solution space behind valid domains. It is often possible to provide a more detailed overview of the solution space that would allow the user to make comparisons between available alternatives faster.

In this section we will illustrate the main idea on two toy examples: *T-Shirt configuration* taken from [HSJ$^+$04], and *Car Configuration* adopted from [AFM02].

Example 1 (T-Shirt Configuration) *Consider specifying a T-shirt by choosing the color (black, white, red, or blue), the size (small, medium, or large) and the print ("Men In Black" - MIB or "Save The Whales" - STW). There are two rules that we have to observe: if we choose the MIB print then the color black has to be chosen as well, and if we choose the small size then the STW print (including a big picture of a whale) cannot be selected as the large picture of a whale does not fit on the small shirt. The configuration problem (X, D, F) of the T-shirt example consists of variables $X = \{x_1, x_2, x_3\}$ representing color, size and print. Variable domains are $D_1 = \{black, white, red, blue\}$, $D_2 = \{small, medium, large\}$, and $D_3 = \{MIB, STW\}$. The two rules translate to $F = \{f_1, f_2\}$, where f_1 is $(x_3 = MIB) \Rightarrow (x_1 = black)$ and f_2 is $(x_3 = STW) \Rightarrow (x_2 \neq small)$. There are $|D_1| \times |D_2| \times |D_3| = 24$ possible assignments. Eleven of these assignments are valid configurations.*

The valid domains for the T-Shirt example are:

color	*size*	*print*
b,r,g,w	s,l,m	mib,stw

They allow 24 assignments. A user is guaranteed that whatever the value she selects, there will be at least one remaining solution. By assigning certain values and evaluating the available options, she effectively explores the entire solution space. For example, after assigning *size=small*, the configurator's response reduces the valid domains to a complete solution:

color	*size*	*print*
b	s	mib

Based on this consequence, a user can change her mind and assign a different value if desired, or perform other decision-support tasks, such as asking for an explanation of why certain values are not available any more, etc.

We argue that it is *unnecessary to hide interactions between variables from a user if the entire solution space can be compactly represented*. All eleven solutions in the above example can be compactly described in only two table rows:

color	*size*	*print*
b,r,g,w	l,m	stw
b	s,l,m	mib

The user is provided with a stronger guarantee: *any combination of values in a row is guaranteed to be a solution*. The user can utilize her perceptual abilities to directly observe the solution space and immediately specify a desired solution. Note how all interactions between variables and values are directly observable, without the need to first assign and explore consequences. In particular, it is easy to see that a small T-Shirt is only available in black and with "Men in Black" print.

Example 2 (Car Configuration) *We are required to configure colors for various parts of a car. There are five variables: $x_1, \ldots, x_5$ representing* bumpers, body, top, doors, *and* hood. *All variables share the same domain $D_1 = \ldots = D_5 = \{white, pink, red, blue\}$. There are two constraints: bumpers and top should have a lighter color than body. Doors and hood must have the same color as the body.*

Initial valid domains in the above example are:

bumpers	*body*	*top*	*doors*	*hood*
w,p,r	p,r,b	w,p,r	p,r,b	p,r,b

They allow 243 solutions, even though there are only 14 solutions. However, the entire solution space can be exactly represented in only three rows:

bumpers	*body*	*top*	*doors*	*hood*
w	p	w	p	p
w,p	r	w,p	r	r
w,p,r	b	w,p,r	b	b

Note how constraints over variables can be directly observed. Body, doors and hood must have the same color. Choices of colors for bumpers and top are mutually independent.

4 Algorithmic Challenges

In this section we formalize the notions related to the quality of visualization and extract the algorithmic challenges associated with providing good visualizations. The Cartesian product of the valid domains can be seen as an *over-approximation* of the solution space Sol:

$$Sol \subseteq VD_1 \times \ldots \times VD_n. \tag{1}$$

This visualization device guarantees backtrack-freeness and completeness by ensuring that each valid domain is a projection of exact solution space onto the corresponding variable, $VD_i = \pi_i(Sol)$ where π denotes a projection operator. In contrast, in the two toy examples we represented the solution space *exactly*, by introducing a *set of Cartesian products*, each product corresponding to a *row* in a *visualization table*:

$$Sol = \bigcup_i D_1^i \times \ldots \times D_n^i. \tag{2}$$

An advantage of such a table over displaying (1) is in a *stronger interaction guarantee*: any combination of values from a single row, $(v_1, \ldots, v_n) \in D_1^i \times \ldots \times D_n^i$ for some row i, is a valid solution. A disadvantage in comparison to (1) is that the number of rows in the table might be too large to represent and for a user to absorb. Hence, the quality of representation (2) is directly related to the number of rows in the table, i.e. the number of Cartesian products in the set. We therefore formulate our first algorithmic challenge as follows:

Problem 1 (Minimal Row Representation) *For a given constraint satisfaction problem (X, D, C), with solution space Sol, what is the minimal number of Cartesian products r_{min} necessary to exactly represent Sol:*

$$Sol = \bigcup_{i=1}^{r_{min}} D_1^i \times \ldots \times D_n^i.$$

Even if we are able to solve the above problem, the required minimal number of Cartesian products r_{min} might be too large for practical use. We suggest, therefore, to relax the requirement for exact representation, and instead to require *a tight over-approximation* that *does not require many rows*. We can always satisfy space requirements by taking a one-row over-approximation (1). On the other hand, we can always achieve desired tightness of the over-approximation by taking the exact solution space representation (2) where each row corresponds to a solution. The associated algorithmic challenge is to achieve an appropriate tradeoff between the two conflicting goals.

Problem 2 (Over Approximation of Rows) *For a given constraint satisfaction problem (X, D, C), with solution space Sol, and for a given maximum number of Cartesian products r_{max} what is the smallest over-approximation Sol^{apx}:*

$$Sol \subseteq Sol^{apx} = \bigcup_{i=1}^{r_{max}} D_1^i \times \ldots \times D_n^i$$

i.e. an over-approximation with the minimum number of elements?

An orthogonal way to improving the Cartesian product representation is to show values for only a subset of variables. Such a subset could be specified either by a user or a system. For example, a user might only care about the values of some "interesting" variables. On the other hand, a system might recognize a subset of "critical" variables whose assignment is sufficient to completely specify the solution. In general, for a given subset of variables $X' \subseteq X$, we want to exactly represent the projection:

$$Sol_{X'} = \bigcup_i \prod_{x_j \in X'} D_j^i. \tag{3}$$

The smaller the number of variables X', the easier it is to represent the solution space $Sol_{X'}$. However, some solutions might become indistinguishable when comparing over a subset of variables only, and the number of identifiable solutions could decrease: $|Sol_{X'}| \leq |Sol|$. As a measure of projection quality we take the number of "lost" solutions: $\delta(X') = |Sol| - |Sol_{X'}|$ and the associated algorithmic challenge is:

Problem 3 (Approximation by Projection) *Given a solution space Sol, and a maximum number of Cartesian products r_{max} what is the subset of variables $X' \subseteq X$ yielding a Cartesian product representation (3) with at most r_{max} rows, such that projection of solution space Sol on X' variables $Sol_{X'}$, involves the maximum number of solutions?*

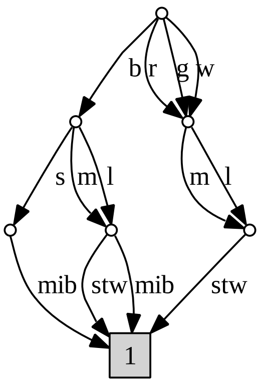
In fact, the research carried in [CO08] already identifies variables that can be ignored *without solution loss*. The authors identify *functionally dependent* variables, whose value is completely specified by assigning remaining *core* variables. Finding a minimum-size core actually helps in addressing the above problem when we insist that the projection must retain *all* solutions. In this work however, we are willing to accept some solution loss $\delta(X')$ if we can further reduce the number of projection variables X' (and therefore Cartesian products r_{max}).

Besides the above methods, a number of other methods could be considered to improve the Cartesian product representation. In particular, we could exploit preferences over variable value pairs. These again could be extracted from a user, or could be inherently present (cheaper price or more memory are always better, everything else being equal). In this paper, we consider *preference shading*, i.e. indicating the quality of a choice for an attribute by shading the table cell with an appropriate color or intensity. If there are more than one value in a cell, an intensity or color is selected based on the most desirable value. While we will not discuss this further in the paper, an example will be shown in Section 6 that uses such a choice.

5 An MDD-Based Approach

In this section we discuss how to address some of the algorithmic challenges mentioned in the previous section, using multi-valued decision diagrams (MDDs) as an underlying representation of the solution space. In particular, we discuss several approaches that minimize the number of rows in a visualization table.

It is important to observe that the MDD allows for the immediate extraction of visual-table representation. Every *meta-path* in the MDD corresponds to a Cartesian product, i.e. a row in the visualization table. For example, an MDD for the T-Shirt example is shown in Figure 5 (left) and the corresponding table extracted from meta-paths is shown in the same figure on the right.



color	*size*	*print*
b	s	mib
b	m,l	mib,stw
r,g,w	m,l	mib,stw

Figure 1. An MDD for the solution space of the T-shirt example. There are three meta-paths in the MDD resulting in the visualization table on the right. Note that this is not the minimal Cartesian product representation.

The generated table has three rows, while the minimal number of rows for the T-Shirt example is two (as illustrated in Section 3). Therefore, the Problem 1 of minimizing the number of rows corresponds directly to minimizing the number of meta-paths in the underlying MDD.

We suggest two techniques for minimizing the number of meta-paths in an MDD. The first technique is *variable reordering*, which is known to dramatically influence the number of MDD nodes. We are, however, interested in variable orders that minimize the number of *paths* rather than nodes. Figure 2 shows an example of how the reordering of the variables can reduce the path-count.

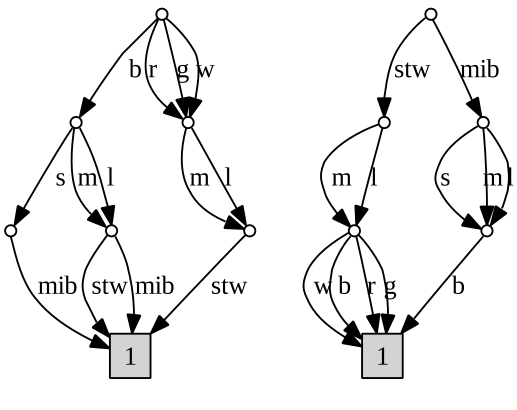


Figure 2. The T-Shirt example with the standard variable ordering (color, size, print) on the left, and with the new variable ordering (print, size,color) on the right. Note that the number of meta-paths reduced from three to two.

While in the T-Shirt example reordering of variables reduced both the number of paths and nodes, note that in general it is not possible to minimize both numbers simultaneously. A variable ordering leading to an MDD with the minimal number of paths is not necessarily the ordering leading to an MDD with the minimal number of nodes. This fact is a generalization of an analogous result that holds for *binary decision diagrams* [Weg00].

The second technique we propose for reducing the number of paths is *non-determinization*. In an MDD, edges (u, u_i, v_i) rooted at the same node u, must have disjoint labels v_i. This an important requirement that guarantees efficiency of a number of important queries. However, for the purpose of visualization, where we care only about the number of paths, this requirement is not necessary. By allowing overlapping values on outgoing edges, we introduce "non-deterministic" choices in the decision diagram and we get a structure that is not an MDD anymore in a standard sense. However, allowing non-determinization can potentially reduce the number of paths as illustrated in Figure 3.

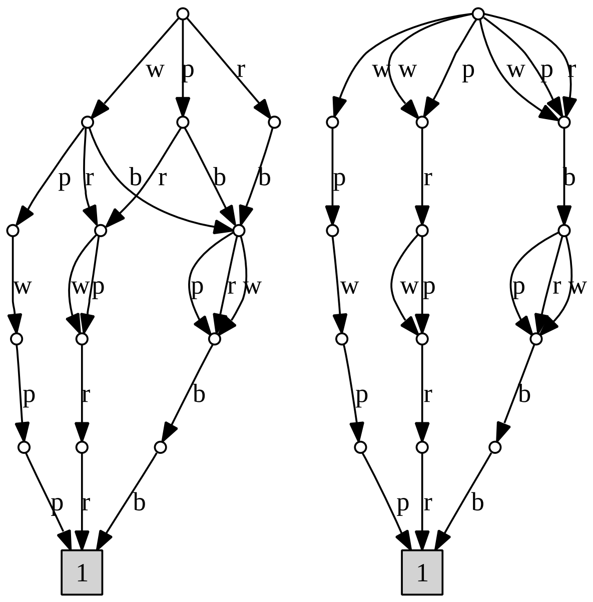


Figure 3. Car configuration example on the left, and after non-determinization on the right. The number of meta-paths is reduced from six to three.

Note that projecting an MDD onto a subset of variables can be efficiently implemented using standard projection techniques for binary decision diagrams. While in theory the size of resulting MDD can be bigger, in practice it is often smaller. While we do not discuss this technique further, we used it in the empirical evaluation in Section 6. Finally, the techniques for approximating the number of rows are currently under development and are part of our future work.

6 Case-Study: Configuring a Digital Camera

In this section we apply the MDD-based techniques of *variable projection*, *non-determinization* and *preference shading* to a real-world configuration benchmark of *Camera* configuration [CO08]. We are given a catalogue of 112 cameras, each characterized by eight attributes: brand, price, resolution, optical zoom strength, flash memory, screen size, thickness and weight. Each of the attributes corresponds to a variable.

In Table 1 we report the effect of variable projection and non-determinization techniques. For smaller subsets of variables X' we compute an MDD M with solution space $Sol_{X'}$. We then apply a heuristic-based non-determinization on MDD M to generate a non-deterministic MDD M_n representing the same solution space $Sol_{X'}$. We report the number of meta-paths P, P_n and nodes $|M|, |M_n|$ for both the initial and non-deterministic MDD respectively.

Table 1. Table illustrating solution loss and row savings by projecting variables for the Camera instance. Column X' indicates indices of variables in the scope of projection. Entry $1-8$ is a shorthand for indicating that all variables $\{x_1, \ldots, x_8\}$ are in the set. Column $|Sol_{X'}|$ indicates the number of solutions in a projected MDD, P_n and P refer to the number of paths in the non-deterministic and initial MDD, while $|M_n|$ and $|M|$ indicate the number of nodes in the non-deterministic and initial MDD.

X'	$\mathbf{\|Sol_{X'}\|}$	$\mathbf{P_n}$	$\mathbf{P}$	$\mathbf{\|M_n\|}$	$\mathbf{\|M\|}$
1-8	112	100	106	388	394
1-7	112	94	103	317	325
1-6	112	87	99	189	200
1-5	112	83	91	143	149
2,3,4,5	111	75	92	116	121
1,2,4,5	110	61	64	79	84
1,2,3,4	109	73	78	78	80
2,4,5	108	49	54	66	65
2,3	91	32	32	34	34
2,5	88	24	24	26	26

We can see that by projecting onto the subset of functionally independent variables [CO08], we have compression without solution loss. For example, projecting onto $x_1, \ldots, x_5$ still yields 112 solutions since the values of variables x_6, x_7, x_8 are completely specified by assignments to $x_1, \ldots, x_5$. Furthermore, we can see that even when we start loosing solutions, we can select projection variables X' in a way that yields a very good tradeoff between solution loss and meta-path reduction. For example, by projecting onto variables x_2, x_4, x_5 we get an MDD with only 49 meta-paths, while loosing only four solutions. it can be also seen that the effect of non-determinization is moderate but still noticeable. Further experimental evaluation is needed to fully understand the potential of this technique.

Based on the results in Table 1, we generated a visualization table for the Camera catalogue projected onto variables x_2, x_4, x_5, i.e. price, zoom strength and flash memory. Furthermore, since all attributes have a natural preference structure (e.g. more/less is better), we implemented a simple preference-shading scheme and generated the same table where all cells are shaded with different intensities of the gray color to indicate the quality of the value. The resulting visualization tables are shown in Figure 4. The entries in the tables are sorted with respect to ascending price. The lighter intensity indicates higher quality. In cells with more than one value, only the best value is shown and "+" or "-" is appended to indicate that there are other values of lesser quality (which are greater or smaller depending on what is considered worse for the attribute).

We can see how shading allows us to immediately spot interesting values. For example, the greatest amount of flash memory (256MB) can be obtained in a middle price range of 299.5\$, while the most expensive camera (675.99\$), in comparison, offers only a small amount of flash memory(16 MB). The same holds for the zoom strength. A high zoom factor 10 is available already starting from 293.99\$. However, it is hard to find both the high flash memory and optical zoom for the same price. Some "lighter table lines" do stand out. For example, a camera with 32MB of flash memory an optical zoom of factor 10 can be purchased for 299.95\$.

7 Conclusions and Future Work

In this paper we argued for moving beyond displaying valid domains as a main form of communicating with a user in interactive configuration. We suggested a richer interaction framework, based on visualizing the solution space through the set of Cartesian products. We proposed several techniques that could improve the Cartesian product representation for larger instances, and evaluated our approach on a configuration benchmark.

In future, we plan to further implement and test the techniques indicated in this paper, as well as develop new techniques for improving Cartesian product representation. In particular, we plan to develop MDD-based techniques for row approximation based on aggregating highly related values.

7.1 Acknowledgments

Tarik Hadzic is supported by an IRCSET/Embark Initiative Postdoctoral Fellowship Scheme. Barry O'Sullivan is supported by Science Foundation Ireland (Grant Number 05/IN/I886).

REFERENCES

[AFM02] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs-application to configuration. *Artificial Intelligence*, 2002.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 1986.

[CO08] Hadrien Cambazard and Barry O'Sullivan. Reformulating table constraints using functional dependencies - an application to explanation generation. *Constraints*, 13(3), April 2008.

[HA06] Tarik Hadzic and Henrik Reif Andersen. A BDD-based Polytime Algorithm for Cost-Bounded Interactive Configuration. In *Proceedings of AAAI'06*, 2006.

[HF92] Paul Hubbe and Eugene Freuder. An efficient cross-product representation of the constraint satisfaction problem search space. In Rina Dechter, editor, *Proceedings of AAAI-92*, AAAI Press, pages 421–427, 1992.

[HSJ+04] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In *PETO Conference*, pages 131–138. DTU-tryk, June 2004.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams.* Society for Industrial and Applied Mathematics (SIAM), 2000.

Price($)	Zoom	Flash (MB)
109.99+	1	16
129.99+	3	16
139.99+	3	12
149.95	3	32-
149.99	4	16
149.99+	3	16-
179.95	3-	16
179.99	3	22
199.95	3	32-
199.99	3	23-
199.99	4-	16
212.99+	6	14
215.99+	3	32
249.95	3	24-
249.99	5	24
249.99	3	32-
293.99+	10	13.4
293.99	3	16
293.99+	3.6	16
299.95+	10	32
299.95	5	32
299.95	3	256-
299.99	10-	16
299.99	5	32,8
299.99	3	32-
319.99	10	10
329.95	3	64
329.99	10	21
329.99	3	20
349.95	3	32-
349.95	1	32
349.99	12-	16
349.99	5	17
375.99	2.4	16
391.99+	12	16
399.95+	12	32
399.95	10	32-
399.95	5.8	32
399.95	3	58-
399.99	12-	16
399.99	4	32-
399.99	3	32-
401.99+	3.5	23
449.99	3	25
499.95	10.7	10
499.99	3	32-
569.99	4	32
599.99	6	32
675.99	10.7	16

Price($)	Zoom	Flash(MB)
109.99+	1	16
129.99+	3	16
139.99+	3	12
149.95	3	32-
149.99	4	16
149.99+	3	16-
179.95	3-	16
179.99	3	22
199.95	3	32-
199.99	3	23-
199.99	1,4	16
212.99+	6	14
219.99+	3	32
249.95	3	24-
249.99	5	24
249.99	3	32-
293.99+	3.6	16
293.99+	10	13.4
293.99	3	16
299.95+	10	32
299.95	5	32
299.95	3	256-
299.99	10-	16
299.99	5	32-
299.99	3	32-
319.99	10	10
329.95	3	64
329.99	10	21
329.99	3	20
349.95	3	32-
349.95	1	32
349.99	12-	16
349.99	5	17
375.99+	2.4	16
391.99+	12	16
399.95+	12	32
399.95	10	32-
399.95	5.8	32
399.95	3	58-
399.99	12-	16
399.99	4	32-
399.99	3	32-
401.99+	3.5	23
449.99	3	25
499.95	10.7	10
499.99	3	32-
569.99	4	32
599.99	6	32
675.99	10.7	16

Figure 4. Both tables represent projection of the Camera catalogue on three variables: price, optical zoom and flash. There are 108 solutions in the table, presented in 49 rows. Corresponding MDD has 66 nodes. The number of rows is reduced from 100 to 49 by loosing only four solutions. If there are more than one value in a cell, only the "most preferred" value is shown, and "+" or "-" is appended to indicate that there are other values of lesser quality. The table shows the results of applying the preference-shading technique on the table on the left.

Towards an association of product configuration with production planning

M. Aldanondo[1], E. Vareilles[1] M. Djefel[1] and Paul Gaborit[1]

Abstract. This communication presents some works relevant to the possibility of coupling together interactive product configuration tools with process planning tools in order to pass decisions made from one to the other. The first section introduces the problem and the general ideas of the proposed solution. Two constraints based models, relevant to product configuration and process planning, are presented. Then first investigations for coupling these two models and associated problems are discussed. An example illustrates our proposal through out the paper.

1. INTRODUCTION

The aim of this communication is to present the first results of a study dealing with the development of an aiding system that will simultaneously allow interactive configuration of a product and interactive planning of its production process. It seems rather logical that on the one hand, product configuration decisions have strong consequences on the planning of its production process and that on the other hand, planning decisions provide hard constraints to product configuration. Therefore, we propose to associate these two problems in order to allow (i) the propagation of the consequences of each product configuration decision toward the planning of its production process and (ii) the propagation of the consequences of each process planning decision towards the product configuration. This should reduce or avoid planning impossibilities due to product configuration and configuration impossibilities due to production planning. This problem originates from a French national funded project called *ATLAS* whose purpose is the development of an open source software able to help industrialists to design (and not only configure) a product and its associated project simultaneously, by passing decisions made from one to the other in order to try to avoid manual repairs.

As far as we know, this kind of problem has not been addressed by the configuration community. In the design community, for more than fifteen years, many works around: Axiomatic Design proposed by Suh in 1990 [1], Design Structure Matrix proposed by Stewart in 1981 [2], or Function Behaviour Structure proposed by Gero [3], have proposed different domains or views (customer, functions, requirements, behaviour, physical, process, resource…) in order to characterize product development. A recent paper from the DSM community [4] proposed a mapping of the four domains: functions, components, process and resources (as shown in figure 1 from [4]). Our propositions are based on these four domains and we associate (i) the function and component views with the product configuration and (ii) the process and resource views with the planning of the production process.

[1] Université de Toulouse – Mines Albi - Centre de Génie Industriel, France, email : {aldanondo, vareilles, djefel, gaborit}@enstimac.fr

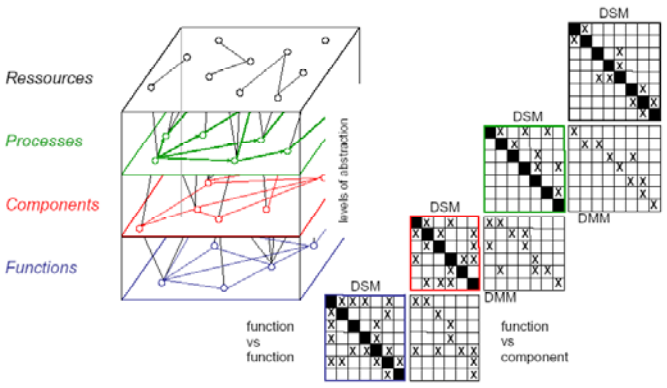


Figure 1. Domains of product configuration and production planning - from the paper (Lindemann U., 2007 [4])

In the configuration community, many authors (among them [5] or [6]) have shown that product configuration could be efficiently modelled and aided when considered as a Constraints Satisfaction Problem (CSP). In a same way, authors interested in scheduling (such [7] or [8]) have shown that project planning could be also modelled and aided when considered as a CSP. A CSP is a triplet {X, D, C} where X is a set of variables, D is a set of finite domains (one for each variable) and C a set of constraints linking the variables [9]. The variables can be either discrete or continuous. The constraints often called compatibility constraints define the possible or forbidden combinations of values for a set of variables thanks to: lists of compatible values, mathematical formulae or comparison operators (>,<,=). Given these elements, we propose to consider simultaneously configuration and planning problems as two constraint satisfaction problems. In order to propagate decision consequences between the two problems, we suggest to link the two constraint based models relevant to configuration and planning as shown in figure 2.

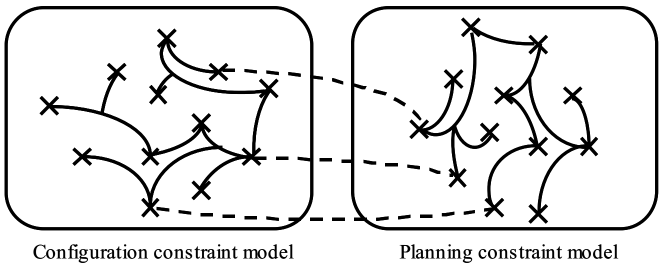


Figure 2. Association of configuration and planning models

The communication is therefore organized as follow. Product configuration is first addressed and planning issues are discussed. The association of the two constraints models are then investigated and discussed. A detailed example run through out the paper.

2 PRODUCT CONFIGURATION

This section concerns the definition of the configuration problem that we address. The problem is first defined, then the constraint model is described with propagation techniques. The description of the example finishes this section.

2.1 Configuration Problem Description.

From previous works achieved concerning configuration, it seems that some common features defining configuration could be:

1. hypothesis: a product is a set of components,
2. given: (i) a generic model of a configurable product able to represent a family of products with all possible variants and options, that gathers (1) a set of component groups (2) a set of product properties and (3) a set of various constraints that restrict possible combinations of components and property values ; and (ii) a set of customer requirements, in which each requirement can be expressed by a selection (or a domain restriction) of a component or a property value,
3. configuring can be defined as "finding at least one component set that satisfies all the constraints and the customer requirements".

We associate previous properties with some kind of product description that matches the “function” domain introduced in section 1. Component and component groups are associated with the physical or component domain of section 1. We consider (for simplicity) that the properties and the component groups have symbolic values, for examples, value of the property “length” can be “2m”, “4m”, “8m”…, value of the component ”engine” can be “E_lp” or ”E_hp”.

2.2 Constraint Model and Propagation Techniques

Each group of components and each product property is associated with a configuration variable. Each component and each property value corresponds with one value of the variable. The constraint represents the allowed (solid lines of figure 3) or excluded (dot lines of figure 3) combinations of components and property values. The Dynamic extension of the CSP, DCSP proposed by [10] and frequently called Conditional-CSP, introduces: (i) Initial variables: variables that exist in any configured product, (ii) Compatibility constraints: equivalent to the CSP constraints defined in [9] and (iii) Activity constraints (arrows on figure 2): allowing to control the variable existence in the following ways: (1) Require: a specified value of a variable "X" implies the existence of variable "Y", (2) Not Require: a specified value of a variable "X" implies the non existence of variable "Y". DCSP allows to modulate the existence of any variable corresponding with a group of components or a product property.

As we target interactive assistance, constraint propagation is obtained thanks to filtering algorithm. As we have reduced the definition domain of the configuration variables to symbolic values, simple arc consistency techniques, defined for discrete CSP and DCSP extension, are used.

2.3 Example relevant to Product Configuration.

The Figure 3 shows the configuration model of our example. The product taken for example is a very simple crane.

In its functional or descriptive view (left part of Figure 3), the product can be defined with four configuration variables:

1. V_length, where “V” stands for vertical, the height of the crane with two possible values 4 and 8 meters,
2. H_length, where “H” stands for hrozontal, the width of the crane with two possible values 2 and 4 meters,
3. M_load, the maximum load with two possible values less than one ton between 1 and 2 tons,
4. Ctr-Cab, modulating the existence of a control cabin with two values “yes” and “no”.

Three constraints reduce the solution space and exclude combinations of values (doted lines between variable values):

1. (V_length = “4m”) incompatible with (H_length = “4m”) a crane cannot have the same height and width
2. (M_load = “ 1t << 2t ”) incompatible with (H_length = “4m”) the maximum load is not compatible with the larger width
3. (V_length = “4m”) incompatible with (Ctr-Cab = “yes”) a height of four meters forbids the presence of a control cabin.

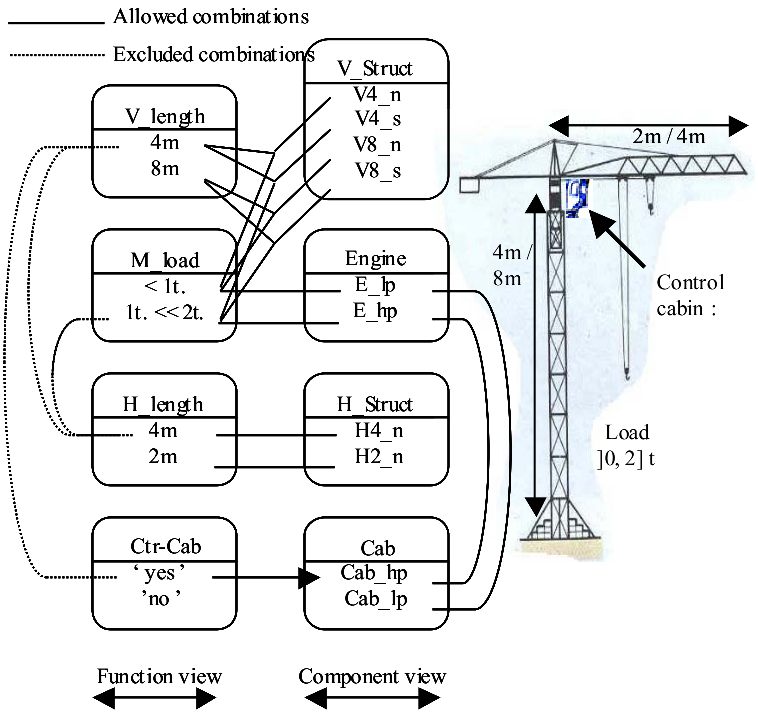


Figure 3. Product configuration model

In its physical or component view (right part of Figure 3) the model shows four groups of components:

1. V_Struct, the vertical structure gathering 4 physical components combining different acceptable loads and lengths: “V4_n”, “V4_s”, “V8_n”, “V8_s”,
2. H_Struct, the horizontal structure gathering 2 physical components according to its length “H2_n”, “H4_n”,
3. Engine, the crane engine gathering 2 physical components with two different power “E_lp”, “E_hp”,

4. Cab, the control cabin gathering 2 physical components according to the engine power "Cab_lp", "Cab_hp".

In this physical view, a single constraint associates the possible engine and the control cabin (solid line between variable values):

1. (Engine="E_lp") compatible (Cab="Cab_lp") and (Engine="E_hp") compatible (Cab="Cab_hp").

The functional and physical views are linked with constraints that show how the components can fulfil the parameters of the functional view. Three constraints show the following possible combinations (solid lines):

1. the height of the crane (V_length) and the maximum load (M_load) are linked with the vertical structure (V_Struct),
2. the maximum load (M_load) impacts the engine selection (Engine),
3. the width of the crane (H_length) is associated with the horizontal structure (H_Struct).

One activity constraint is necessary to allow triggering the existence of the variable (Cab) corresponding with the component group "cab".

3 PRODUCTION PLANNING

This section is concerned by the definition of the planning problem we address. The problem is first defined, then the constraint model is described and propagation techniques are discussed. The description of the example finishes this section.

3.1 Planning Problem Description.

As we address production planning, we will consider a task entity and that the production process is a set of task entities. As the present time we only consider planning with infinite resource capacity. A task entity is defined with:

1. temporal parameters: possible start time (pst), possible finishing time (pft), possible duration (pdt),
2. resource parameters: required resource (rrs), quantity of required resource (qrs).
3. compatibility constraints can link possible duration (pdt), with required resource (rrs) and/or quantity of required resource (qrs).

The three temporal parameters are numerically defined with intervals while resource parameters can remain symbolic (as we consider infinite resource capacity). The production process is defined with:

1. a set of task,
2. a set of precedence constraints between task expressing that task Y is after task X or: Y.pst > X.pft These kinds of constraints have certain similarities to the Allen's primitives [11] (before, after, starts, finishes . . .).

With these elements the production process can gather sequential tasks and parallel tasks as shown with the "AND" node (&) in the left part of figure 4. Planning decisions can correspond with:

1. temporal parameters, value selection or domain restriction,
2. resource and/or resource quantity selection or domain restriction .

In order to be able to select a path or a branch in the process (OR node in the right part of figure 4) it is necessary to be able to control the existence of the task entities.

Interactive planning corresponds with the propagation of all the constraints in order to reduce the definition domain of all temporal parameters, resource parameters and to select the path in the case of "XOR" nodes.

3.2 Constraint Model and Propagation Techniques

A task entity (TI, for task "I") is defined and gathers the three real variables defined with intervals corresponding with the three temporal parameters:

1. pst: possible start time (TI.pst),
2. pdt: possible duration (TI.pdt),
3. pft: possible finishing time (TI.pft).

When all the variables are numerical and each constraint written with a mathematical formula, $f(x_1, x_2…x_n)$ (= > <) 0, Bound Consistency, proposed by [12] and based on interval arithmetic [13], proposes filtering techniques that operate fine if:

1. $f(x_1, x_2…x_n) = 0$ can be projected on any variable x_i meaning that a function f_i exists as: $x_i = f_i(x_1,x_2…x_{i-1}, x_{i+1},…x_n)$,
2. any projections f_i is continuous and monotonous,
3. only one constraint expressed as a formula acts on a same variable subset. Bound Consistency is weak when more than one constraint acts on a same sub-set of variables (corresponding to some constraint intersection). It is shown in [14] that a simple problem gathering two variables and three constraints cannot be fully filtered,
4. each variable occurs only on time in a formula. In the opposite case, for example: $x_1^2 – x_1 - x_2 = 0$, it has been shown that the way to express the mathematical expression, for example: $x_1^2 – x_1 - x_2 = 0$ or $x_1*(x_1 – 1) - x_2 = 0$, influences the quality of the filtering operation [14].

As we consider only the two following kinds of constraints:

1. TY.pst > TX.fst expressing that task Y is after task X
2. TI.pft = TI.pst + TI.pdt expressing the relation between starting, finishing times and duration, for any task "I",

we can use Bound consistency filtering algorithm with the various projections of the two previous kinds of constraints. Planning works perfectly as far as there is no "XOR" node.

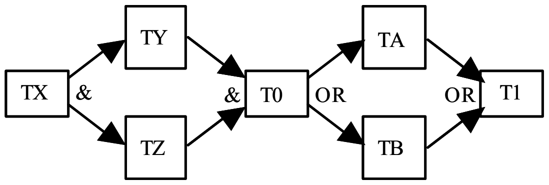


Figure 4. Production process model

In order to deal with the "XOR" node or to modulate the existence of some tasks or sets of tasks, it is necessary to be able to express that some of them have their existence conditioned. A CSP extension called Conditional and Composite Temporal Constraint Satisfaction Problems or CCTCSP has been proposed in [15]. But, CCTCSP does not allow the tasks (represented by events) to have an interval for their length of time. We have therefore define a Meta-task entity "XOR" that gathers two tasks that are linked with a "XOR" node.

Assume that we have a process T0 then TA or TB then T1 (as in the right part of figure 4):

1. TA with TA.pst, TA.pdt, TA.pft
2. TB with TB.pst, TB.pdt, TB.pft

In order to modulate the existence of TA and TB we add:

1. the single numerical value 0 in the definition domain of the possible duration of TA and TB, TA.pst, TB.pst,
2. a constraint that excludes two null values or two non null values for the duration of TA and TB, admitted tuples are therefore: (TA.pdt = 0 , TB.pdt ≠ 0) and (TA.pdt ≠ 0, TB.pdt = 0).

The possible starting and finishing times of the XOR_AB meta task gathering TA and TB, XOR_AB.pst and XOR_AB.pft, are constrained by:

1. XOR_AB.pst > T0.pft meta task starts after T0
2. XOR_AB.pst ≤ TA.pst meta task starts before TA
3. XOR_AB.pst ≤ TB.pst meta task starts before TB
4. TA.pft ≤ XOR_AB.pft meta task finishes after TA
5. TB.pft ≤ XOR_AB.pft meta task finishes after TB
6. T1.pst > XOR_AB.pft meta task finishes before T1

The possible duration of the XOR_AB meta task is constrained by:

1. XOR_AB.pdt = TA.pdt U TB.pdt, the possible duration of the meta task equals the union of the possible duration of tasks A and B
2. XOR_AB.pdt > 0, the possible duration of the meta task is always strictly positive.

The meta-task "XOR" temporal parameters are propagated with the same Bound consistency filtering algorithm. When two sequences of tasks are considered, it is first necessary to aggregate each branch then to define the meta-task "XOR" on the aggregated task.

3.3 Example relevant to Production Planning.

The Figure 5 shows the planning model relevant to the production of the crane example.

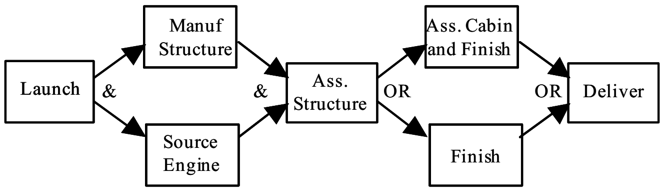


Figure 5. Production process model of the crane.

The following tasks are therefore considered. Only possible duration (pst) for each task is provided. A required resource with a quantity is define just for two tasks (*Manuf Structure* and *Deliver*). is defined. A launching task has been added:

1. Launch (L): event that allows to set a launching time,
   L.pst = L.pdt = L.pft = 0,
2. Manuf structure (MS) corresponds with the manufacturing of the two structures (V_Struct and H_Struct), a small and a large machine can be used with a quantity 1,
   MS.pdt = [3 , 6],
   MS.rrs = {large_mach , small_mach}
   MS.qrs = "1"
3. Source Engine (SE): corresponds with the sourcing of the engine (E_lp and E_hp),
   SE.pdt = [2 , 4],
4. Ass Structure (AS): assembly of the two structures with the engine, duration always equal to 2,
   AS.pdt = 2,
5. Ass Cabin and finish (ACF): if a cabin is present in the configuration,
   ACF.pdt = {0 , [3 , 4]}
6. Finish (F): if no cabin in the configuration,
   F.pdt = {0 , [1 , 2]}
7. XOR_ACFF.pdt= ACF.pdt U F.pdt, and XOR_ACFF.pdt > 0,
   XOR_ACFF.pdt= { [1 , 2] , [3,4] }
8. Deliver (D): corresponds with the delivering of the completed crane. Two different transportation resources can be used and modulate duration.
   D.pdt = [1 , 2]
   D.rrs = {fast_transp , slow_transp}
   D.qrs = "1"
   Constraint (D.rrs , D.pdt): allowed combinations:
   { (slow_transp, [1.5 , 2]) , = (fast_transp, [1 , 1.5]) }

These elements are summarized in figure 6. All starting and finishing dates have an initial interval value [0 , 20]. All constraints are propagated and the domain of all temporal parameters are reduced and provide the result of figure 6.

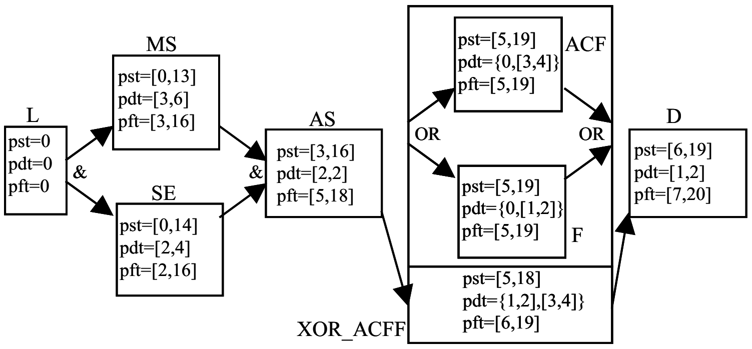


Figure 6. Production process after first propagation.

4 COUPLING CONFIGURATION AND PLANNING

Coupling constraints are introduced followed by some illustrations with the crane example.

4.1 Coupling Constraints

A coupling constraint is a compatibility constraint that links a variable of the configuration model with a variable of the planning model. Various kinds of coupling constraint have been identified. Any variable of the configuration model, belonging either to the function or the component view, can belong to a coupling a constraint. On the planning model side, three cases can be identified:

1. the planning variable is a resource parameter (rrs or qrs). This allows to propagate the impact of a configuration decision on the selection of the required resource and/or resource quantity, reverse behaviour from resource selection to product configuration is also possible,
2. the planning variable is the temporal parameter duration (pst), that does not belong to a XOR meta task. This allows to propagate the impact of a configuration decision on the modulation of the duration of a task, reverse behaviour from duration modulation to product configuration is also possible,
3. when the temporal parameter duration (pst) belong to a XOR meta task, the previous behaviour is completed with the possibility to select one of the task of the meta-task.

4.2 Illustration with the Crane Example

Four coupling constraints are first proposed and then coupling is illustrated.

The three cases of coupling constraints are illustrated as follow:

1. the planning variable is a resource parameter:
   1 - The resource of the task *manufacturing structure*, MS.rrs, is linked with the height of the crane, V_Length, with the following allowed combinations, Constraint (V_Length, MS.rrs): { ("4m", "small_mach") , ("8m", "large_mach") }
2. the planning variable is a duration parameter that does not belong to a XOR meta task:
   1 - The duration of the task *manufacturing structure*, MS.pdt, is linked with the maximum load, M_load, with the following allowed combinations : Constraint (M_load, MS.pdt) :
   { ("<1t", [3 , 4.5]) , ("1t << 2t", [4.5 , 6]) }
   2 - The duration of the task *source engine*, SE.pdt, is linked with the engine component, Engine, with the following allowed combinations : Constraint (Engine, SE.pdt) :
   { ("E_lp", [2, 3]) , ("E_hp", [3, 4]) }
3. the planning variable is a duration parameter that belong to a XOR meta task in order to select the existence of a task :
   1 - The duration of the task *assemble cabin and finish*, ACF.pdt, is linked with the existence of a control cabin, Ctr-Cab, with the following allowed combinations : Constraint (Ctr-Cab, ACF.pdt) : { ("no", [0]) , ("yes", [3 , 4]) }

4.2.1 Propagating product configuration decisions toward production planning.

If we assume the following configuration decisions:

1. V_length = "8m"
2. H_length = "2m"
3. M_load = "1t << 2t"

and leave open the decision relevant to the existence of a control cabin. The configuration provide the component set :

1. V_Struct = "V8_n"
2. H_Struct = "H2_n"
3. Engine = "E_hp".

Coupling constraints provide :
The selection of resource :

1. MS.rrs = "large_mach"

The modulation of the task duration :

1. MS.pdt = [4.5 ,6]
2. SE.pdt = [3, 4]

Planning constraint propagation provide (also shown in figure 7):

1. Launch (L) : L.pst = L.pdt = L.pft = 0,
2. Manuf structure (MS) : MS.pdt = [4.5 ,6]
   MS.pst = [0, 11.5] and MS.pft = [4.5 , 16]
3. Source Engine (SE) : SE.pdt = [3 ,4]
   MS.pst = [0, 13] and MS.pft = [3 , 16]
4. Ass Structure (AS)) : AS.pdt = [2 ,2]
   AS.pst = [4.5, 16] and AS.pft = [6.5 , 18]
5. Ass Cabin and finish (ACF) : ACF.pdt = {0 , [3 , 4]}
   ACF.pst = [6.5, 19] and ACF.pft = [6.5 , 19]
6. Finish (F) : F.pdt = {0 , [1 , 2]}
   F.pst = [6.5, 19] and F.pft = [6.5 , 19]
7. XOR_ACFF.pdt = { [1 , 2] , [3,4] }
   XOR_ACFF.pst = [6.5, 18]
   XOR_ACFF..pft = [7.5 , 19]
8. Deliver (D) : D.pdt = [1, 2]
   D.pst = [7.5, 19] and D.pft = [8.5 , 20]

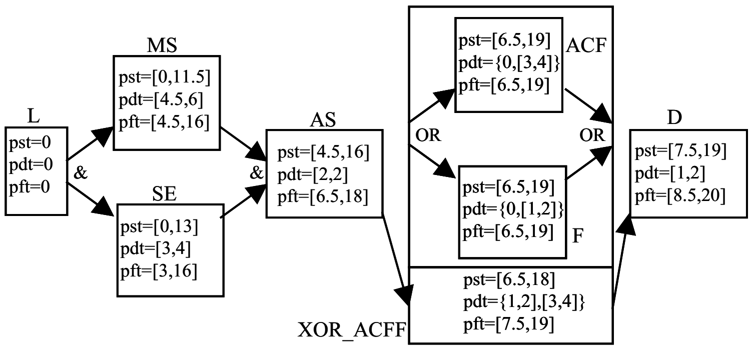


Figure 7. Planning after configuration.

One of the consequence of the coupling lies in the increase of the total finishing time (D.pft) from [7 , 20] to [8.5 , 20]. If we assume now that the person in charge of planning wants to secure the manufacturing tasks and selects the maximum duration for them : MS.pft = [6,6] and SE.pft = [4,4]. Planning propagation gives the planning of figure 8 with a total finishing time D.pft = [10,20].

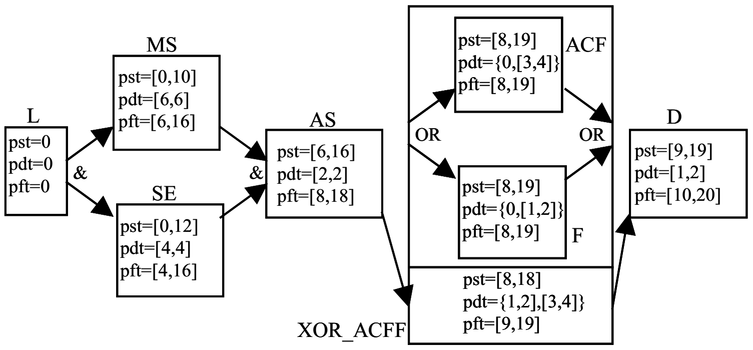


Figure 8. Planning after Manufacturing freezing duration.

4.2.2 Propagating production planning decisions toward product configuration.

If we assume that the total finishing time D.pft should be now less than 11. This implies :
For task D :

1. D.pft = [10 ,11] and D.pst = [9 ,10],

For meta task XOR_ACFF, task F and ACF :

1. XOR_ACFF.pft = [9 ,10] and XOR_ACFF.D.pst = [8 ,9],
2. F.pft = [8 ,10] and F.pft = [8 ,10]
3. ACF.pft = [8 ,10] and ACF.pft = [8 ,10]

and the propagation of the constraints expressing that finish time equal start time plus duration (Tpft = T.pst + Tpdt) and the constraint of the XOR node give :

1. Task ACF : ACF.pdt = [0,0]
2. Task F : ACF.pdt = [1,2]
3. meta task XOR_ACFF : XOR_ACFF.pdt = [1,2]

Planning propagation has therefore selected task F and canceled task ACF. The null value for ACF.pdt is then propagated to the configuration model and forbid the selection of a control cabin by reducing the variable Ctr-Cab to the value "no".

5 CONCLUSIONS

The aim of this communication was to present the first results and prospective ideas about the development of an interactive aiding system, that simultaneously allows product configuration and production planning. The main interest and goal of this system is to be able to take into account :

1. product configuration decisions when dealing with production planning,
2. production planning decisions when dealing with product configuration.

We have first presented our problem and proposed to associate each problem with a constraint satisfaction problem and to link the two problems with coupling constraints.
Then, a simple configuration problem was defined and modelled with two views : functions and components. Constraint propagation was achieved thanks to arc consistency techniques. An example dealing with a crane was described.
The planning problem was defined and modelled thanks to a network of tasks that allows "AND" and "OR" nodes. Bound consistency techniques were used to propagate constraints. The crane example was extended with production planning entities.
The final section described the constraints that can associate the two previous models in order to pass decisions made from one to the other. The crane example was used to show (i) how configuration decisions were propagated to planning and (ii) how planning decisions were propagated to configuration.

One of the interests of the proposed system is that it relies only on the simple assembly of two constraint filtering techniques : arc consistency for discrete CSP and Bound consistency for numerical or mixed CSP, that can be soon consulted at http://cofiade.enstimac.fr/cgi-bin/cofiade.pl (chose model ECAI08-crane). The other main interest is relevant to the application domain, where the possibility to manage interactions between product configuration and production planning has been addressed. These elements must be considered as primary results that need to be consolidated with structured (or multi-level) planning, finite resource capacity planning and less routine design.

ACKNOWLEDGMENT

The authors would like to thank :

1. the French National Research Agency (ANR) and the 7th Strategic Activity Domain (Architecture and Integration) of Aerospace Valley for their involvement in this project,
2. their partners in the ATLAS project : Anyware Technologies, Pulsar Innovations, Sigma Plus and LAAS-CNRS from Toulouse (France), LGP-ENIT from Tarbes, (France), and IMS-LAPS from Bordeaux (France).

6 REFERENCES

[1] Suh N., 1990, The principes of design , Oxford Series.
[2] Steward, Donald The Design Structure System: A Method for Managing the Design of Complex Systems" IEEE Transactions on Engineering Management, vol. 28, pp. 71-74, 1981
[3] Gero J.S., 1990, Design prototypes : a knowledge representation schema for design, AI magazine, Vol 11 n°4, p 26-36
[4] Lindemann U., 2007, A vision to overcome "chaotic" design for X processes in early phases, Int Conference on Engineering Design (ICED), Paris France.
[5] Sabin D., Weigel R., "Product Configuration Frameworks – A survey", *IEEE Intelligent Systems*, vol. 13, n° 4, 1998, p. 42-49.
[6] Soininen T., Tiihonen T., Männistö T., Sulonen R., "Towards a General Ontology of Configuration", *AIEDAM,* vol. 12, n° 4, 1998, p. 357-372.
[7] R. Dechter, I. Meiri, J. Pearl, *Temporal Constraint Satisfaction Problems*, Artificial Intelligence, 49, pp. 61-95, 1991.
[8] P. Laborie. 'Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results'. *Artificial Intelligence*, 143, pp. 151-188. 2003.
[9] E. Tsang, *Foundations of constraints satisfaction*, Academic Press, London, 1993.
[10] Mittal S., Falkenhainer B., « Dynamic Constraint Satisfaction Problems », *Proceedings of the 9th National Conference on Artificial Intelligence AAAI*, Boston, USA, 1990, p. 25-32.
[11] J. Allen, *Maintening knowledge about temporal intervals*, Communication of the CACM, tome 26(11), pp. 832-843, 1983.
[12] [Lhomme 1993] O. Lhomme - Consistency techniques for numerical CSPs - IJCAI 93, Chambéry, France, 1993.
[13] R.E Moore - Intervals Analysis - Prentice Hall, 1966.
[14] O. Lhomme and M.Rueher - Application des techniques CSP au raisonnement sur les intervalles - Revue d'intelligence artificielle, Dunod, Vol. 11, pp 283-311, 1997.
[15] M. Mouhoub and A. Sukpan, A New Temporal CSP Framework Handling Composite Variables and Activity Constraints, the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05) pp. 143-149, Hong-Kong, November 14-16, 2005.

Debugging Structure-based Configuration Models

Thorsten Krebs [1]

Abstract. Modern product manufacturers face the problem of managing a variety of product types that change over time. To avoid time-consuming redesign and manual adaptation, complex products are assembled from a given set of smaller components. Configuration is a well-known approach to support the composition of products from a given set of components. In an environment where configurable components and product types continuously evolve, overview of the represented knowledge can get easily lost. This paper introduces a product-centered framework that semantically distinguishes the conceptual representations of products and components within a configuration model for improving the reasoning about impacts that changes to the configuration knowledge have. It further presents three typical use cases that are of interest for companies that configure continuously evolving product types.

1 Introduction

Modern product manufacturers face the problem of managing a variety of product types that change over time. Each *product type* offers alternative and optional choices from which a customer can choose. Such a variety of products is typically needed to satisfy customers with different demands. In order to stay competitive, product manufacturers need to diversify the product variety and improve existing product types over time. To avoid time-consuming redesign and manual adaptation, complex products are assembled from a given set of smaller components. This helps to decrease the effort of production and to increase the performance and functionality of products [21]. The application of configuration tools is a trend to reach this goal.

Configuration is a well-known approach to support the composition of products from a given set of components. *Structure-based configuration*, in particular, employs hierarchical specialization structures and composition structures in a *configuration model*. Within a configuration model, all potentially configurable products of a domain are implicitly represented by defining the components from which a product can be composed, component attributes and relations between the components. Structure-based configuration models are especially well-suited to represent products that are assembled from a given set of smaller components. The focus on composition structures enables a top-down configuration approach and improves handling product domains with high combinatorial nature.

Objective Typical configuration domains, like consumer electronics, cars, PC's, complex machines or software applications often consist of several hundreds or thousands of components and restrictions on how the components can be combined. In an environment where the configurable components and product types continuously evolve, overview of the represented knowledge can easily get lost. We will use the car domain as an illustrative example throughout the remainder of this paper.

When changing a product type in a way that some component is no longer required, a situation may emerge in which component representations or constraints become superfluous. For example, when a car manufacturer retires his only car with a diesel engine, all representations of diesel engines and related constraints unnecessarily blow up the configuration model's complexity. Another source of irritation is the difficulty to overview and manage both conceptual representations and constraints. Interactions between both kinds of representations may cause situations in which a concept representing a component that is intended to be configured for a product type is in fact *inconsistent*, i.e. no instance of this concept can be created during product configuration. This may happen, for example, when our car manufacturer introduces a new on-board computer and is not aware of a constraint restricting components that consume too much energy, as the computer does.

This paper introduces a product-centered framework that semantically distinguishes the conceptual representations of products and components within a configuration model. The distinction improves the reasoning about impacts that changes to the configuration knowledge have and allows the knowledge engineer to focus on the product domain rather than on knowledge representation. We present three typical use cases that are of interest for companies that configure continuously evolving product types: (1) which components are relevant for configuring which product type(s) and (2) which are not relevant for any product type, as well as (3) which of the relevant components can indeed be instantiated during product configuration.

Reader's Guide The remainder of this paper is organized as follows. Section 2 introduces basic modeling facilities on which the presented use cases rely. Sections 3, 4 and 5 discuss the three use cases and describe practical algorithms. Section 6 presents related work and finally Section 7 summarizes the paper.

2 Knowledge Representation

The work presented in this paper uses a Description-Logic based knowledge representation similar to the one proposed in [15]. Differences are the selection of supported constructors and that we use the *Semantic Web Rule Language* (*SWRL*[2]) [10] to represent rules (called constraints in this work).

More formally, we use the the knowledge representation language SWRL-$\mathcal{ALCQI}$+$(\mathcal{D})$, that consists of the $\mathcal{ALCQI}$+$(\mathcal{D})$ component and the SWRL component. Additionally to [15] we allow inverse roles for reasoning from filler to role specifier, transitive closures

[1] HITeC e.V. c/o University of Hamburg, Germany, email: krebs@informatik.uni-hamburg.de

[2] *SWRL* is the proposed standard for specifying rules in future releases of the *Web Ontology Language* (*OWL*).

over roles for reasoning about whole subtrees, quantified number restrictions for restricting the type of filler and concrete domains for representing attribute values. Due to space limitations we omit a definition of the used constructors and refer the interested read to [1].

Please note that reasoning in a hybrid approach consisting of a decidable DL and a decidable rule component may not be a decidable problem. However, we can trade in a little expressivity for decidability: the combination of any Description Logic with *DL-safe rules* is proven to be decidable [18]. DL-safe rules allow classes and properties from the Description Logic component to appear freely in the antecedent or consequent of the rule, with the only restriction being that they can be applied only to explicitly named instances [17].

Modeling Facilities A *concept* is a description which gathers common features of a set of objects in the domain. Concepts are not objects (and therefore not individuals) but rather patterns that are instantiated (as individuals) during a configuration process. Concepts are modeled containing two different hierarchical relationships: specialization and composition.

Specializations are used to model a generalization / specialization hierarchy, also called *taxonomy*. Every concept has exactly one ancestor, if not the root concept, and can have an arbitrary number of descendants. Hence, the taxonomy is a tree structure. Children of the same parent are also called *siblings*.

Compositions form a composition hierarchy, also called *partonomy*. Concepts are either *primitive* (sometimes called *atomic*) or *composite*. This means that they reside at the leaves of the composition hierarchy or are the root of a subgraph, respectively.

Attributes define characteristics of concepts and are represented with a name and a value. The name is uniquely identifiable within the taxonomy. The value is restricted to a set of pre-defined value domains, such as integer numbers, real numbers, ranges of the two, strings, and sets of all three.

Properties, i.e. both attributes and composition relations, are inherited from a parent to all its children. Properties can be *refined* for concepts that reside at lower levels of the taxonomy. Values of refined properties are more specific than the original value.

Instances are instance of exactly one concept and inherit all properties, i.e. both attributes and composition relations, from this concept definition. The important property of instances is that they have an identity, which allows them to be distinguished from one another and to be counted. This means that instances with different names, even with the same description, will be different instances.

Interdependencies and restrictions between concept definitions are expressed with *constraints*. Constraints represent non-hierarchical dependencies between concepts and concept properties, as well as the existence of of certain concept instances. A constraint definition consists of an antecedent and a consequent. The antecedent specifies a *pattern*, consisting of a conceptual structure, that evaluates to true whenever the pattern matches an instance structure. The consequent is executed when the pattern evaluates to true [6]. Constraints are unidirectional: a constraint's consequent needs to be satisfied if and only if its antecedent evaluates to true.

Modeling Products Product types are represented with the above described modeling facilities. Basically, the representation of a product type is a subtree of the configuration model. This means that one concept is specified as the root of that subtree (the concept representing the product itself) and a number of additional concepts, related via composition relations, represent parts from which actual products of that type can be assembled.

Abstract and Concrete Components The taxonomic hierarchy contains *abstract concepts* and *concrete concepts*, i.e. generic concepts used for taxonomically grouping similar component types and concepts representing specific components that can actually be assembled for realizing a product, respectively. For example, a gasoline engine is an abstract concept, while the 320ccm-sports-engine is a concrete concept: there is an engine of this type on stock. Concrete concepts in this sense are leaf concepts with fully specified property values, i.e. atomic values for attributes and composition cardinalities. Such a restriction is reasonable due to the fact that even only slightly different components, for example the same component in different colors, typically are assigned unique identifiers (e.g. names or numbers) for identifying them on stock.

We assume that a product may specify abstract concepts as parts. This means that during the configuration process instances of abstract concepts are created and specialized either directly by selecting one of the children or indirectly according to their properties that are incrementally refined. Hence, we assume it is the goal to specialize every concept instance to be instance of a concrete concept.

3 Which Components Are Relevant for a Product Type?

This use case emerges because typically multiple product types are represented within a single configuration model. Reasoning about a single product type based on the whole configuration model, it may happen that a component is constrained inconsistently in two ways: required for one product type and excluded for another one. For example, while the luxury edition of a car requires a navigation system or computer, small and cheap cars have no space for these components. It is thus important to rely on information relevant for the concerned product type only.

The concepts that represent components required for providing the functionality of a product type constitute a segment of the configuration model. A *segment* in this sense is a subtree of the configuration model containing all information relevant for one or more product types, depending on what the segment is used for. The segmentation algorithm starts with the corresponding product type's overall composite concept and creates an extract of the part concepts defined within the partonomy. Other concepts that are related to the part concepts are identified by following the taxonomic relations, partonomic relations and constraints.

Please note that the algorithm is inspired by [19]. It also traverses upwards and downwards in the taxonomy but also needs to address concept properties, i.e. both attributes and composition relations, and constraints. The segmentation algorithm is extended accordingly.

Downwards Traversal of the Partonomy The first concepts to include in the segment are the parts of the overall composite, then parts of the parts and so on, until all leaf concepts of the partonomy are reached. Parts of a concept are especially important because they represent components of which the product can be assembled. All part concepts that are specified with a maximum cardinality greater than zero are included in the segment. This means that optional parts (with cardinality $[0; 1]$) are also included, they can be selected by a customer and are required to provide the product type's functionality.

Comparing Attribute Values Product types may specify product-specific property values, i.e. restricted values for properties of part concepts. Simply including part concepts into the segment would include more than what is actually relevant for the product type. In

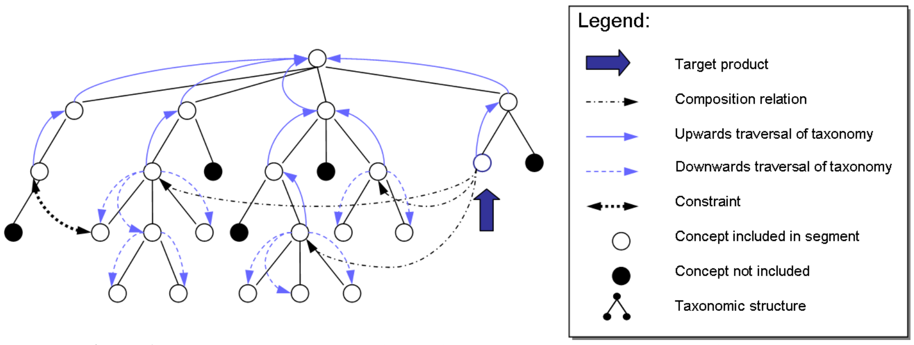


Figure 1. Creating the segment of a configuration model that is relevant for a product type.

order to include only concepts that specify properties with allowed values we use two filters. The first filter is to omit concepts with at least one property value that does not have a common subset with the restricted value. The second filter is to use the common subset of original value and restricted value for all properties of concepts that are included in the product-related segment. For example, for a car that specifies a gasoline engine with at least 200ccm as a part, the corresponding concept's attribute value is restricted accordingly and all children with less than 200ccm are omitted.

Upwards Traversal of the Taxonomy For every concept included in the segment, the parent of that concept is included, then the partent's parent and so on, until the root concept is reached. Parents of concepts are important due to the fact that they provide critical information about inherited properties. One might think of merging superconcepts in order to yield a taxonomy less deep but this would destroy some of the semantic accuracy as we will see later on.

Downwards Traversal of the Taxonomy The child concepts of all parts that were reached by traversing the partonomy are also included, then the children's children and so on, until the leaf concepts of the taxonomy are reached. Children of parts are especially important because it is the leaf concepts of the taxonomy that represent concrete components to be assembled in a product.

The algorithm does not include children of the concepts included by upwards traversal of the taxonomy. They are not relevant for the product type and doing so would result in including the entire taxonomy. This is something one definitely wants to avoid.

Sibling Concepts in the Taxonomy Sibling concepts of the product type's part concepts and of the part's parents are not included into the segment. They are not relevant for the concerned product type due to the fact that specializations of a concept represent alternatives and one alternative is determined by the composition relation specifying the part concept. All other alternatives can safely be discarded, unless they are specified as a part of the considered product type on a different composition path. For a car with a gasoline engine, for example, the diesel engine or other siblings are irrelevant. Please note that this holds for the product type's part concepts and their parents only; for children of the product type's part concepts all siblings are relevant and included into the segment.

Constraints For every concept that is now included in the segment the algorithm needs to check whether the concept itself or any of its properties are constrained. Constrained concepts can easily be identified due to the fact that they appear within the constraint definition. When this is the case, all other concepts that appear within the constraint definition need to be included in the segment as well for being able to check the constraint's satisfiability. Concerning all concepts of a constraint definition becomes important, for example, when not all concepts directly belong to the prodcut specification but represent external influence like legal regulations, emission laws, or other context like humidity and temperature constraints on hardware components. Additionally, parents and children of the constrained concepts are included by traversing upwards and downwards in the taxonomy.

Complexity Figure 1 illustrates the segmentation algorithm. Starting from the overall composite, the algorithm first traverses the partonomy, then the taxonomy upwards to the root concept and downwards to the leaf concepts, starting from already included concepts. After that, constraints that constrain included concepts are analyzed to include all concepts related via constraints. From these constraint-related concepts, the taxonomy is also traversed upwards and downwards to include all parents and children.

Let the configuration model contain n concept definitions in total, including the root concept and the $k < n$ represented components. The number of part concepts of the product type $l \leq k$, the number of levels within the taxonomy $i < k$, the maximum number of children in a taxonomic level $j < k$ as well as the number of constraints m and their maximum arity a are considered the influential parameters of this algorithm. We know that $i \cdot j \leq n$ because the taxonomy cannot contain more concepts than the total number of concepts. The worst case computational complexity is $O(n^2 + (a - 1)m)$ and thus polynomial.

4 Which Components Are Not Relevant for Any Product Type?

For identifying components that are not relevant for any product type the segmentation algorithm from the previous section can simply be executed for every product type. Doing so yields in a segment of the configuration model containing all concepts that are relevant for at least one product type. Vice versa, components that are represented by concepts not included in this segment are not relevant for any product type.

Concepts that are not relevant for any product type can be considered superfluous. Superfluous concepts have a different status than concepts that are relevant for at least one product type and this should be reflected in the configuration model. When the configuration model defines all components that are produced and held on stock, for example, superfluous concepts lead to unnecessary production and should be removed or made otherwise inaccessible.

However, there are various reasons why superfluous concepts may emerge. For example, the concept may have been relevant for some product type that was retired in the meantime. Another reason may be that a knowledge engineer models components first and uses them to define new product types afterwards. A product manufacturer may also be interested in supporting his products beyond the time they were sold and use a versioning mechanism for keeping information about old product types and components. Thus, superfluous concepts need to be carefully analyzed and should not be removed right away. A possibility to manage such concepts is to differentiate between concepts that are needed, intended to be needed, obsolete (but maybe still needed because of versioning and product maintenance support) and indeed superfluous.

Complexity Executing the segmentation algorithm for one product type is evaluated in the previous section. When executing this algorithm for all p product types, p is another influential parameter of the algorithm: executing the segmentation algorithm for p product types yields a computational complexity that is p times the previous computational complexity. The worst case computational complexity is $O(p \cdot (n^2 + (a-1)m))$ and thus still polynomial.

5 Which Components Are Reachable?

Creating a product-related segment of the configuration model, we know which components are relevant for a product type. But not all concepts that represent relevant components can indeed be reached during product configuration. *Reachability* of a concept describes the fact that it can be instantiated when configuring a product of a given type and includes two aspects. The first aspect is the fact that concepts are modeled in a taxonomy and all superconcepts of a product type's part concept can not be instantiated, unless otherwise specified as a part of this product type. The second aspect is the fact that constraints may rule out instances of certain concepts directly (specialization-related and composition-related constraints) or indirectly (attribute-related constraints). The fact that both aspects need to be evaluated for deciding on concept reachability stems from the hybrid representation combining concepts and constraints.

Please note that evaluating reachability of a concept only produces reliable results when executed on a segment that is relevant for a single product type. A segment that is relevant for multiple product types may contain contradictions, like a constraint ruling out concept instances for one product type that are required for another one.

5.1 Taxonomy-based Reachability

Leaf concepts represent concrete components that can be assembled in a product. When configuring a product, it is tried to specialize instances of more general concepts until they are instance of a leaf concept. Hence, all concepts that are higher in the taxonomy than the product type's part concepts are not reachable during product configuration. For a car with a gasoline engine, for example, the generic motor cannot be instantiated when configuring this type of car. The generic motor would, among others, also include the diesel engine, which is not admissible. We assume that the product type's part concepts and their children are, or at some point in time were, *intended to be used* for the concerned product type.

Complexity Let again the configuration model contain n concept definitions in total, including the root concept and the $k < n$ represented components. The number of part concepts of the product type $l \leq k$, the number of levels within the taxonomy $i < k$ and the maximum number of children in a taxonomic level $j < k$, for which we know that $i \cdot j \leq k$, are considered the influential parameters of this algorithm. The worst case computational complexity of identifying all concepts that are reachable with respect to the partonomy is $O(k^2)$ and thus polynomial.

5.2 Constraint-based Reachability

Not all concepts representing components that are *intended to be used* for a product type are indeed reachable. There may be constraints directly ruling out instances of specific concepts or indirectly restricting property values of concepts in a way that they are inconsistent, i.e. no instances of these concepts are admissible.

Constraints are defined on the conceptual level but are evaluated on the instance level. Hence, constraints are evaluated only when an existing instance structure matches the conceptual structure of the constraint's antecedent. The obvious way to evaluate constraint satisfaction for a conceptual model is instantiating all leaf concepts and evaluating the resulting constraint net for every admissible combination of these instances. However, starting from the leaf concepts will most probably lead to generating instance combinations that are consistent with respect to the product type's partonomy but are actually inconsistent with respect to constraints defined for concepts higher in the taxonomy. Operating top-down prevents from testing inconsistent instance combinations. We start by creating instances of the product type's part concepts and specialize them along the taxonomy down to the leaf concepts while evaluating the emerging constraints that are defined for the corresponding concepts.

A special case of the top-down approach is when no constraints are defined for any of the generic concepts and their subconcepts. In this case all leaf concepts are theoretically reachable. But there is the obvious restriction that an instance of a concept can only be specialized to one of its children. Thus, all leaf concepts that are subconcepts of the same concept are disjoint and instances of siblings can only coexist when more than one part is instantiated.

Specialization-related Constraints restrict the potential choice between children of the constrained concept by (dis)allowing the specialization to one or more of the children. This means that all siblings but the ones that remain consistent are not reachable and can be discarded for further constraint evaluation. For example, when a given motor requires a specific gear shift, this can be modelled by specializing the generic gear shift, that is specified as a part concept of the product type, to the required one.

Composition-related Constraints restrict the number of instrances that may be generated for part concepts of a specific composition relation. But so long as the maximum cardinality of the composition relation is greater than zero this does not affect the set of reachable concepts. Only when the maximum cardinality of a composition relation is set to zero, then an instance of the specified part concept is no longer admissible. This concept and its children are not reachable and can be discarded for further constraint evaluation. Requiring a component, for example, can be modelled by setting the minimum cardinality to one. Excluding a component analogously can be modelled by setting the maximum cardinality to zero.

Attribute-related Constraints rule out values of attributes and may cause inconsistency: a concept becomes inconsistent when the value of an attribute is constrained in a way that none of its values are consistent with respect to the constraint. Inconsistent concepts and their children are not reachable and can be discarded for further constraint evaluation. For example, tires and rims both have a size and a width attribute that apparently have to match for being able to mount a tire on the rim.

5.3 Constraint Satisfaction

A *constraint network* consists of a finite set of *variables* (i.e. the constrained concepts, concept attributes or composition relations), with respective *domains* (i.e. sets of concept names, concrete domains or cardinality intervals, respectively) which list the possible values for each variable, and a set of *constraints* that are defined on a subset of the variables. The *arity* of a constraint depends on the number of variables. A *unary constraint* is defined on a single variable, a *binary constraint* is defined on two variables, and so on.

It is a well-known fact that constraint satisfaction problems (CSPs) belong to the class of NP-complete problems. There exist a number of so-called *network consistency algorithms* (e.g. *node consistency*, *arc consistency*, *k-consistency*) that have a polynomial computation time, but do not solve a CSP completely [14, 12]. These consistency algorithms can be seen as *approximations* in that they impose *necessary*, but not *necessary and sufficient* conditions on the existence of a solution to a CSP.[3]

Of course, approximations are not sufficient when trying to find out which leaf concepts are indeed reachable during product configuration. But what we can do is try to reduce computational complexity by getting as far as possible from the worst case!

Node Consistency as Preprocessor There may be unary specialization-related, composition-related and attribute-related constraints that can be solved by a simple node consistency algorithm. In this sense, evaluating node consistency for unary constraints plays the role of a preprocessor for subsequently solving the constraint net. The preprocessor eliminates local inconsistencies that would otherwise later be stumbled upon. For example, specializing the generic gear shift to a more special one, constraints that are defined for other but this gear shift need not be checked due to the fact that no instances will match their antecedents.

Reducing the Search Space Specialization constraints and composition constraints rule out instances of certain concepts. When these constraints are evaluated first, fewer instances of leaf concepts have to be taken into account when solving the constraint net afterwards. Additionally, often value ranges are provided for which not all values are actually covered by leaf concepts. In order to reduce the domain size of constraint variables, the attribute values of all leaf concepts can be merged into a common superset for the attribute of a constrained concepts that is higher in the taxonomy. For example, the size attribute of the tire concept may specify an interval of $[16; 20]$ inch, but in fact there are only concrete tires with 16, 17 and 20 inch. Using the common superset $\{16; 17; 20\}$ allows less combinations with the sizes of rims and saves computation time. Additionally, we assume that leaf concepts represent concrete components that have finite value domains for all attributes. Hence, by merging attribute values of leaf concepts to a common superset, infinite value domains need not be considered.

When the value domain of an attribute is modified by solving the constraint net, some of the leaf concepts become inconsistent due to the fact that the concerned attribute cannot contain values that are not covered by at least one of the corresponding attributes of the leaf concepts. Inconsistent concepts, as well as their children are not reachable and can be pruned from the search space.

Evaluating Constraints on the Conceptual Level The constraint net need not be evaluated for every potential combination of leaf concept instances. It can rather be evaluated for instances of those concepts for which the constraint is specified. When evaluating the constraint for instances of the parent concepts the constraint net is evaluated once instead of for every combination of child concepts. Let us assume there are 10 different tires and 5 different rims. For the corresponding constraint there are 50 value combinations that need to be checked. For the generic tire and rim concepts the constraint needs to be checked only once. When solving a constraint modifies attribute values, some of the leaf concepts become inconsistent and can be pruned analogous to the previous step.

In case there are constraints defined for children of the considered concept, the algorithm traverses downwards in the taxonomy and evaluates the emerging constraints. For concepts that allow one instance within the product configuration, like the gear shift, only one of its children can be asserted simultaneously. Other children, however, can also be constrained. This means that the algorithm needs to branch and create separated constraint nets for evaluating consistency of the sibling concepts.

Newly emerging constraints can be evaluated locally first: in case the constraint is unary, only one variable is addressed, while in case the constraint is higher-ary, other variables need to be concerned. Local constraint evaluation is sufficient, except for two cases. In the first case the value domain of the starting variable has changed and this variable is connected to other constraints as well. In the second case the constraint includes further variables, whose value domains have changed, and they are connected to other constraints. In both cases the constraint net needs to be evaluated. For all but these two cases, local constraint evaluation saves a considerable amount of computation time.

Independent Constraint Subnets The last paragraph introduces a valuable improvement for the constraint satisfaction algorithm. When evaluating a product type's consistency, the complete constraint net needs to be evaluated once. But this constraint net may actually consist of multiple smaller subnets that are mutually independent. The constraint net consists of mutually independent subnets when the subnets do not share any variables, which is the case, for example, when the size attributes of tire and rim are constrained to be equal and a radio requires a speakers set. Evaluating consistency of leaf concepts, only constraints that emerge because the algorithm traverses downwards in the taxonomy and dependent constraints need to be evaluated. Other constraints, that are independent from the emerging constraints, can be ignored at this point: either they were already evaluated when the algorithm addressed higher levels of the taxonomy, or they are irrelevant for the currently considered combination of concept instances.

Complexity Constraint satisfaction problems belong to the class of NP-complete problems and it is not possible to present an algo-

[3] For an in-depth overview of constraint processing we refer the interested reader to [2].

rithm with polynomial worst-case complexity. But we have pointed out several aspects that – in most cases – considerably reduce computational complexity: node consistency as a preprocessor, reduction of the search space, evaluation of constraints on the conceptual level, and mutually independent constraint subnets.

6 Related Work

Configuration tools have been developed in the last decades. Two structure-based configuration tools that employ a very similar representation of configuration models are *KONWERK* [7] and *EngCon* [9]. No model editor is publically available so that configuration models have to be created and maintained manually, e.g. using a text editor, which is error-prone, of course.

There is a number of readily available ontology editors, e.g. [5], and Description Logic reasoners, e.g. [20, 8]. Practically all ontology editors focus on the *Web Ontology Language* (*OWL*)[4]. There are differences in expressivity between OWL and our conceptual language, mainly because of constraint support. Although hybrid frameworks combining Description Logics and Logic Programming, such as *CARIN* [13] or $\mathcal{AL}$-log [3], have been proposed they still are a well-known research problem [16] and no ready-to-use solution is available.

[4] introduce consistency-based diagnosis of configuration models that is based on first-order logic for representing configuration knowledge. The goals of that work and the work presented in this paper are very similar, but rely on different knowledge representations. To the knowledge of the author, no prior approach for managing knowledge about evolving products is based on structure-based configuration representation mechanisms. The product-centered knowledge management approach is novel in the sense that it semantically distinguishes concepts representing products and concepts representing components from which products can be assembled. This distinction improves the reasoning about impacts that changes to the configuration knowledge have. In this respect the approach supports the typical work of a knowledge engineer and enables to focus on the product domain rather than on knowledge representation.

7 Summary

This paper introduces a framework for analyzing the representations of configurable components that can be assembled for realizing a product and representations of the product types themselves. We discuss three typical use cases that are of interest for product manufacturers that use structure-based configuration and present practical algorithms for all three use cases. The first two use cases, i.e. evaluating which components are relevant for which product type(s) and which components are not relevant for any product type, are simple and have a linear worst-time complexity. The third use case, i.e. evaluating which of the relevant components can indeed be reached during product configuration, has non-polynomial worst-time complexity due to the fact that it comprises constraint satisfaction. But we point out several aspects that – in most cases – considerably reduce computational complexity.

The work presented in this paper belongs to a larger framework for knowledge management supporting the evolution of configurable products [11]. This framework not only considers analyzing the representations of configurable components and the product types but also defines evolution processes that directly deal with impacts that changing components, e.g. introducing new components, new versions or variants, or retiring components, have on product types and, vice versa, impacts that changes to product types have on the configurable components.

[4] http://www.w3.org/TR/owl-ref/

REFERENCES

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.

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

[3] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf, 'AL-log: integrating datalog and description logics', *Journal of Intelligent and Cooperative Information Systems*, **10**, 227–252, (1998).

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

[5] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu, 'The evolution of protégé: an environment for knowledge-based systems development', *International Journal of Human-Computer Studies*, **58**(1), 89–123, (2003).

[6] Peter M.D. Gray, Suzanne M. Embury, Kit Y. Hui, and Graham J.L. Kemp, 'The evolving role of constraints in the functional data model', *Journal of Intelligent Information Systems*, **12**(2-3), 113–137, (1999).

[7] Andreas Günter and Lothar Hotz, 'KONWERK - a domain independent configuration tool', in *Proceedings of Configuration (AAAI Workshop)*, pp. 10–19, Orlando, FL, USA, (1999). AAAI Press.

[8] Volker Haarslev and Ralf Möller, 'Racer system description', in *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, pp. 701–705, Siena, Italy, (2001). Springer Verlag.

[9] Oliver Hollmann, Thomas Wagner, and Andreas Günter, 'EngCon: A flexible domain-independent configuration engine', in *Proceedings Configuration (ECAI 2000-Workshop)*, pp. 94–96, (2000).

[10] Ian Horrocks and P. F. Patel-Schneider, 'A proposal for an OWL rules language', in *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, pp. 723–731, New York, NY, USA, (May 17-22 2004). ACM Press.

[11] Thorsten Krebs, 'Kowledge management for evolution of configurable products', in *Twenty-seventh SGAI International Conference on Artificial Intelligence (AI-2007)*, Cambridge, England, (December 2007). Springer Verlag.

[12] Vipin Kumar, 'Algorithms for constraint-satisfaction problems: A survey', *AI Magazine*, **13**(1), 32–44, (1992).

[13] Alon Y. Levy and Marie-Christine Rousset, 'Combining horn rules and description logics in CARIN', *Artificial Intelligence*, **104**(1-2), 165–209, (1998).

[14] Alan K. Mackworth, J. A. Mulder, and W. S. Havens, 'Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems', *Computational Intelligence*, **1**(3), 118–126, (1985).

[15] Deborah L. McGuinness and Jon R. Wright, 'Conceptual modelling for configuration: A description logic-based approach', *Artificial Intelligence Engineering Design, Analysis and Manufacturing*, **12**(4), 333–344, (1998).

[16] Boris Motik and Riccardo Rosati, 'Closing semantic web ontologies', Technical report, University of Manchester, UK, (2007).

[17] Boris Motik, Ulrike Sattler, and Rudi Studer, 'Query answering for owl-dl with rules', *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, **3**(1), 41–60, (2005).

[18] Riccardo Rosati, 'On the decidability and complexity of integrating ontologies and rules', *Journal of Web Semantics*, **3**(1), 61–73, (2005).

[19] Julian Seidenberg and Alan Rector, 'Techniques for segmenting large description logic ontologies', in *Ontology Management: Searching, Selection, Ranking, and Segmentation (workshop at 3rd International Conference on Knowledge Capture (K-Cap) 2005)*, p. 4956, Banff, Canada, (2005).

[20] Evren Sirin, Bijan Parsia, Bernardo C. Grau, Aditya Kalyanpur, and Yarden Katz, 'Pellet: a practical owl-dl reasoner', *Web Semantics: Science, Services and Agents on the World Wide Web*, **5**(2), 51–53, (2007).

[21] Markus Stumptner, 'An overview of knowledge-based configuration', *AI Communications*, **10**(2), 111–126, (1997).

What Makes Product Configuration Viable in a Business?

Albert Haag[1]

Abstract. Product configuration capabilities are gaining in business importance as technology enables highly individualized specification and handling products and services. However, its pervasion in the market (while increasing) is slower than might be expected. What are the prerequisites for successfully enabling such functionality? What are the impediments? I argue that the following issues must be addressed:

1. The total cost of implementing product configuration must be substantially lowered, e.g. by
   - A better match between technology and the problem. This might be achieved by a clearer stratification/ definition of the problems in business terms, from "engineer to order" to "pre-stocked variants", say.
   - Simplifying deployment and integration of configurator technology with other involved components such as CAD systems
   - Making the problems themselves more tractable by business re-engineering if necessary
2. End to end integration with the associated business processes.
3. Inclusion of the surrounding environment into a product model inasmuch as this imposes constraints on the valid configurations

As an outlook, I want to suggest some examples of applications that might become important in the future.

1 Preamble

One general caveat upfront: this paper would benefit by more exact numbers indicating the actual economic performance of product configuration. Unfortunately, this must be deferred to a later extended version. Here I aim to simply communicate perceptions that result from more than 20 years working in this field, first at The Battelle Institute (in Frankfurt), and (starting in 1992) at SAP AG, and provoke thought than to guarantee historical accuracy.

2 Expectations and Reality

Product configuration was one of the R&D topics supported by substantial funding of artificial intelligence in the 1980s throughout the world, but particularly in Europe. I participated (then as an employee of Battelle) in one such project [1]. The goal was to create a state-of-the-art tool for planning and configuration (PLAKON) and to apply this to several commercially relevant applications. The project charter cited a foreseen enormous economic potential of such applications. Surprisingly, these expectations have not yet been fully met.

A lot has been achieved, of course. The number of people and companies earning their livelihood in the field is now quite substantial[2]. SAP itself has between 1500 and 2000 customers (each having many users) that employ product configuration in some form as part of the sales and logistics processes.

However, product configuration is still often stigmatized as being difficult, risky and costly. While some projects that yield an obvious benefit have been publicized, many customers do not yet perceive product configuration as a reliable means to generate demonstrable return-on-investment. This is because it is sometimes difficult to calculate the benefits vs. the cost.

There are in my opinion two reasons for this:

1. Costs for maintaining the product models, the software maintenance, and overall integration are perceived as high (with some justification). These costs are readily visible on the balance sheet of a company.
2. The benefits of being able to offer configurable products is sometimes, but not always apparent on the balance sheet. Before the advent of sales through the internet a knowledgeable sales engineer was often able to personally handle customer specifications without a configurator. Abstract calculations of potential gains in efficiency of the salesforce by deploying configurator support were offset by non-acceptance of such systems by these users.

The internet has changed this to a certain extent. An end-customer ordering directly will perceive an advantage in being able to customize the product they are buying. Between two competitors both offering similar products, the one with the better configuration capabilities (from a user interaction point of view) may have a decisive competitive advantage. Thus, if anyone succeeds in providing configuration capabilities for a particular product market (segment), this would change the perception of the benefits obtained though configuration more or less immediately.

3 Examples of Business Scenarios

3.1 Ordering a Set of Components

A scenario with fairly simple business integration is creating a configuration that simply determines a list of components that will be later used to assemble the configuration at the customer's site. A typical example of this is designing a custom kitchen[3] to be comprised only of predesigned components listed in a catalog. The product model for the sales configurator needs

[1] SAP AG, Germany, email: albert.haag@sap.com

[2] One indicator is the evolving presence of the field in the internet. Searching the internet (Google) with the term "product configuration" now yields over 400.000 hits and around 10 sponsored links of tool vendors. Interestingly searching with the term "mass customization" yields only 360.000 hits and no sponsored links. Searching with both terms yields 23.200 hits. Curiously, in WikiPedia (English) there is no entry for "product configuration" but one for "mass customization".

[3] Other examples would be configuring a model train or designing the cabin layout of an airliner (one of the applications considered for PLAKON).

to know the components in the catalog, their properties, price, and availability. Constraints between these components that should hold in a valid solution need be known to the configurator but need not be known in the back-end fulfillment system.

On the other hand, this scenario usually requires some form of interactive visualization. This poses additional requirements for the sales configurator and its integration with a visualization tool. As a minimum the configurator must produce

1. the list of components needed
2. One or more CAD-like drawings that allow an unambiguous assembly of the system at the customer's site.

3.2 Make to Specification

A scenario with more complex business integration requirements is creating a configuration to be manufactured in-house. A typical example of this is buying a car. Typically the following distinct phases are involved

1. The sales-person or the end-consumer specifies the properties of the car. This step is an interactive sales-configuration
2. The resulting sales configuration is translated into input to the manufacturing process. Usually, additional details that are not relevant to the sales-configuration must be derived non-interactively. For this reason this phase is referred to as configuration completion.
3. The completed configuration is used to derive all needed components and routings (manufacturing steps)

A price for the product will often directly result from the sales-configuration in itself. Costs and availability, however, may require completing all three steps.

The product model needs to fulfill several consistency requirements with the master data used in the logistics processes:

1. It must be ensured that the sales-configuration produces only configurations that can be consistently fulfilled. I.e. there should be no hitches in the two non-interactive steps.
2. It must be ensured that sales-configuration actually sells the products logistics has planned for. (E.g. if production is oriented to produce mainly red cars, it is a disaster if no red cars are sold, because the configurator does not offer this choice.)
3. The description of the sales-configuration must be available for inclusion in the invoice.

It is not unusual for data to be changed at all ends at all times. Changes on the logistics side tend to be more frequent than at the sales side (except for price changes).

3.3 Categorizing Business Scenarios

In my opinion, it is one of the open challenges to provide a definitive list of scenarios and problem categories. The terms assemble-to-order, make-to-order, engineer-to-order are too coarse to effectively categorize the problem and mean different things to different people.

Other factors that play a role in problem classification:

1. Size of the configuration (how many properties and constraints are there)
2. Single-level or multi-level (does the configuration have configurable components like the custom kitchen)?
3. Is the fulfillment process completely automated or are there manual steps

4. Factors that Contribute to Success

An ROI for product configuration can only be clearly estimated if two conditions are met:

1. The scenario to be implemented and with it all issues that must be addressed in the affected business processes must be identified upfront. The ROI can only be estimated if solutions for all these issues are readily apparent, Pragmatically simplifying the approach may significantly improve confidence and chances of success.
2. The costs for maintaining the configurator environment (s.a. maintaining the product model, data for any associated functions like CAD or pricing, etc.) need to be estimable with confidence. This is usually only the case if similar solutions that can be used as a reference exist.

4.1 Simplify the Scenario

Both the configurator vendor and their customer have an interest in a pragmatic approach to the problem at hand. When selling a personal computer, for example, there are two extremal approaches (simple vs. complex from the business perspective):

1. Predefining (and perhaps pre-building and pre-packaging) the possible variants.
2. Allowing a very low-level construction of the computer system from a set of potential components

4.1 Integrate the Master Data

Maintenance of the product model and keeping this consistent and in synch with the master data used in the business processes is a known challenge that often results in high maintenance costs. Managing the expectations in this area and having a clear solution for this is important for the continued success of a product configuration solution.

4.2 Full-Fledged User-Experience

The capabilities of the configurator itself can contribute significantly to the success. In terms of performance, explanations, conflict handling, etc.

5. Outlook

Configuration is an enabling technology. It can provide a competitive advantage on the sales side, and it provides efficiency gains on the manufacturing side as a means of dealing with complexity. But it may also open up new lines of business. Instead of just enabling selling and producing new products (or services) A combination of products and services might also be used in other ways:

1. Repair/ recycling of products - currently it is often cheaper to by a new product than to repair an existing one.
2. More highly configurable personal transportation - Providing solutions with a better mix of public and private transportation would make the former more viable.
3. Configuration of financial products based on individual needs analysis
4. Purchasing-by-specification. Low volume production (such as producing quilts, say, or harvesting individual trees) is currently handled by informal manual interaction. Means of collaborating by collaborating on specifications might open new markets here.

REFERENCES

[1] R. Cunis, A. Guenther, H. Strecker(Eds.): Das PLAKON-Buch, Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen. Informatik-Fachberichte 266 Springer 1991.