AVIATION
2008
12(4): 124–128

ABSTRACT STRUCTURE OF CONFIGURATION SYSTEM FOR PRODUCT DATA MANAGEMENT SYSTEM

Linas Burneika

Vilnius Gediminas Technical University, Faculty of Mechanics. J. Basanavičiaus 28, Vilnius, Lithuania.
E-mail: linas@mail.lt

Received 30 April 2008; accepted 01 December 2008

Linas BURNEIKA
Research interests: efficient product data and configuration management, Satisfiability Modulo Theories (SMT) solvers.
Present position: PhD student at the Faculty of Mechanics, Vilnius Gediminas Technical University, Lithuania.

Abstract. A product configurator is a system providing questions and sets of possible answers about products with sets of possible answers. This paper proposes a new idea for a configurator in which information about products are expressed as a structured net of interconnected classes of different types. Classes hold information about assemblies, product structural links, and logical constraints. Some classes have references to technical data on a product data management (PDM) system. Such a system allows flexible representation of knowledge about product configurations for production of aviation equipment.

Keywords: product configuration management, product data management, knowledge representation.

1. Introduction

The problem of configuration emerged as a research topic in the 1980s as the result of manufacturing companies going from mass production to mass customization. Configurators not only solve the mass-customization paradigm, but also are among the most successful applications of artificial intelligence technology. In the product configuration process, a user interactively makes adjustments to the product (an industrial machine, an aircraft, modular aviation equipment, etc.) according his specific needs, using supporting software called a configurator. It calculates a specific product variant that fulfils requirements and meets technical and non-technical constraints. Choices for each available component are usually modelled as variables over finite domains, and information about product specifications is encoded as propositional constraints over these variables.

A trend can be observed that the economy of the 21st century will be based on highly specialized solution providers. The management of the configuration of constantly evolving and changing products must therefore be easily supported, preferably by ordinary engineers without advanced IT skills. While configuration of standardized, mostly well defined products can be quite well achieved, the case for complex and short lifecycle products or services is still an open research issue. This work is intended to address this case.

2. Related works

There is a long history in the research and development of configuration tools in knowledge-based systems. The first attempts were made by introducing rule-based systems. Later research progressed to the development of higher level representation formalisms, such as various forms of constraint satisfaction problems

ISSN1648-7788 print / ISSN 1822-4180 online
http://www.aviation.vgtu.lt
DOI: 10.3846/1648-7788. 2008.12. 124-128

(CSP) or description logics (Soininen *et al.* 1999, McGuinness *et al.* 1998).

Configuration tasks are more dynamic in nature and therefore a representation of a CSP, in which all variables of a problem must be known from the beginning, is not appropriate in many application In order to solve this, the dynamic constraint satisfaction was developed (Soininen *et al.* 1999). It is more suitable for representing and solving configuration tasks because the set of variables of a problem may vary according to some activity constraints. By A. Felfernig and M. Yokoo a distributed dynamic CSP is defined, and a modification of the asynchronous backtracking algorithm from R. Dechter is applied for problem solving (Felfernig *et al.* 2001, Yokoo 2001, Dechter 1990). The limitations of a dynamic CSP representation become evident when configuring large technical systems. To overcome this problem, a generic CSP representation has been proposed (Fleischanderl *et al.* 1998). There, new instances of problem variables can be created from meta-variables during problem solving.

In CSP approaches, configuration can exploit powerful constraint problem solvers to solve complex problems (Freuder *et al.* 2003, Gottlob *et al.* 2000, Beuche *et al.* 2004). The alternative symbolic approach has to split computations to an offline and an online phase (Hadzic *et al* 2004). They first compile valid user assignments to efficient data structures, such as reduced ordered binary decision diagrams (Subbarayan *et al*. 2004). If the compiled representation is small enough, then the already available efficient algorithms deliver basic configuration functionalities.

In general, CSP and symbolic approaches provide good performance parameters, but considering the user-friendliness requirement for a configurator, one can see that both approaches have representation formalism that is difficult to comprehend (Subbarayan *et al*. 2004). Although commercial configurators have a user interface layer, they still do not solve numerous problems arising during frequent production updates. Another problem is that the process of constructing a configurable product requires additional skills and is very different from product design tasks.

The ideas for a product configurator proposed in this paper relate to previous research projects such as (Zhang *et al.* 2006, Magro *et al*. 2001). They are looking for possible ways to configure problem decomposition to improve the efficiency of solving the problem and apply methods of parallel computing. The approach proposed in this work evolves from similar concepts; the representation of configurational knowledge is decomposed according to product structure. Representation formalism is not symbolic, but object driven, however. The task of this paper is to express information about product configurations as a net of interconnected objects, further called the configuration model. The objects of the model are interlinked with respect to product structure and have references to engineering data on the PDM system. This scheme makes the configuration model easy for engineers to understand and provides flexibility for making changes in configurations.

3. Novelty of the work

The product configuration model defined in this paper falls into the engineer-to-order category. Contrary to similar works, the configuration model is designed to allow fast addition of new variants for a product and modifications of existing variants. The architecture of the model includes close relations with the PDM system – a source of engineering data for building new product configurations. The PDM system is also a place where implicit bills of materials are stored after product configuration is completed. Another original feature of the model is its object-driven approach. This gives several benefits: parsing or translating of symbolic syntax existing in conventional configurations is eliminated and classes express knowledge about products. It is easier to understand for end users because classes resemble conventional bills of materials.

4. The configuration model

The product configuration model is constructed using concepts of object-oriented modelling and classification of components. The aim of the configuration model is to provide a method for compact description of multiple product variants. The structure of the model consists of an abstract layer and an implementation layer. This paper is dedicated to the abstract layer only, and the implementation details are not discussed. The abstract layer defines abstract assembly, property, and constraint classes. These classes are construction elements for building a definition of a configurable product. Each class has a specific functionality useful for expressing particular aspects of product variants.

5. An abstract generic class

For the manufacturing process, explicit BOMs (bill of material) of product assemblies are necessary. If multiple variants of a single assembly exist because of customatization, they are typically represented by separate BOMs for every variant. This is not convenient and efficient if the number of variants is large.

To streamline product documentation, it is necessary to find a way to express BOM information differently. In this work, a definition of abstract generic class is introduced. It is a generalization of all assembly variants, and it is the core entity of the configuration model. For better clarity, abstract generic class further is called class in this article (Fig 1).

Class can be used to represent both parts and assemblies, but the main purpose of the class is to represent the structure of a real assembly. A class may therefore have an array of references to other classes that are either subassemblies or parts of the product. Every reference contains quantity and other attributes (units of measurement, notes etc.) of the class.

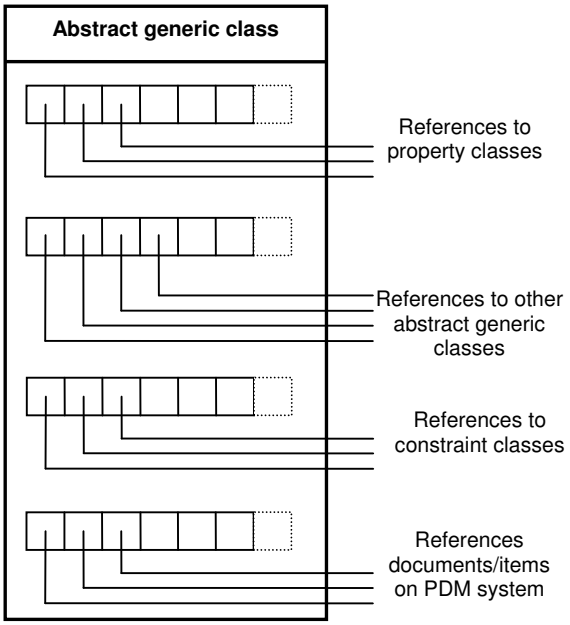

Fig 1. Structure of the abstract generic class

The class also has references to other specific classes (properties, constraints) Classes of properties and constrains are used to create conditions that determine whether the class is present in the structure of a product or not.

If the class represents a component of a part, it will definitely have references to documents (usually a 3D model and drawing) in the PDM system. If however the class is an assembly, it may also refer to the PDM for auxiliary documents. When product configuration is complete, these references can be used to create the explicit BOM for the variant of the product.

6. Inheritance of abstract generic classes

Generalization of assembly is achieved by deriving classes. Initially one parent class that includes only a subset of assembly information common to all variants needs to be created. Then new classes are derived from the parent class. Each new class inherits everything from its parent class, but users of the configurator can refine these classes by adding more specific information. There can be as many inheritance levels as necessary.

For example, in figure 2, class A defines a generic assembly that always contains at least two components another class B and component a2. Two new classes are derived from base class A. Each derived class includes additional information. Class A1 has extended the For example, in figure 2, class A defines a generic assembly that always contains at least two components: another class B and component a2. Two new classes are derived from base class A. Each derived class includes additional information. Class A1 has extended the assembly structure with a new component, a3, while class A2 overrides inherited component a2 with a4. Classes A1.1 and A1.2 are inherited from A1 for the addition of even more detail into the product structure, but for clarification this is not shown in the figure. Every class existing at a leave node of the inheritance tree represents a particular configuration of a root class. In this example, class A has three configurations. A class B has two classes derived, making two configurations, B1 and B2, of assembly B. It follows that the product defined by class A has a total of six configurations.

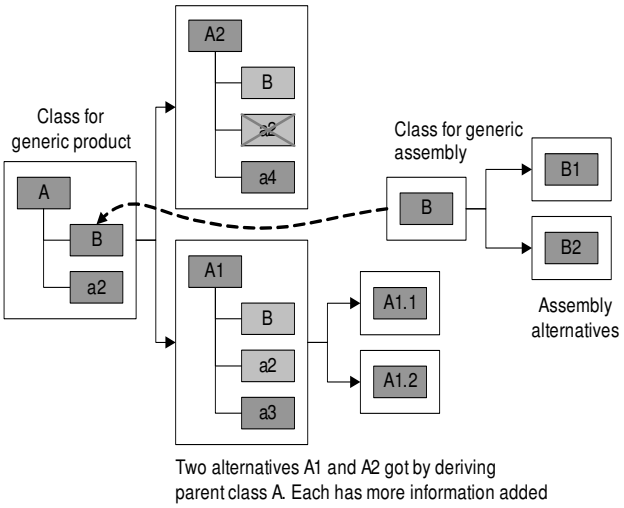

Fig 2. Inheritance example of generic classes

Experience shows that in most cases new product variants are designed by finding an existing product that most closely fulfils the customer's requirements and modifying it. New design efforts are therefore minimized by heavily reusing previous work. The principle of inheritance provides an easy way to add new variants to the configuration model while preserving existing variants.

Choices for product configuration can be easily determined from the model. In the example, the inheritance tree of class A is a question itself with three possible choices for the user: class A2, A1.1, and A1.2. The tree of class B provides two choices. During the configuration process of this example product, the user has to answer two questions in order to get a fully defined assembly of class A.

7. Property class

Property classes are an important part of the configuration model and are intended to define non-geometrical information of parts and assemblies. A property class has a finite list of possible values. Examples of properties, if they are attached to the class representing the part, are paint colour, surface texture, typographical elements, and sometimes – even material. Properties are also useful for assembly structure control. For example the property "design type", containing three values – "standard", "prestige" and "deluxe" – may influence which components must be present in the BOM after configuration. To achieve this control however, particular components have to be connected to property values through constraint classes.

8. Constraint class

A lot of information about product variants is represented by inheritance of abstract generic classes, but in some cases this generalization is not complete. In reality complex interdependencies between components may exist. For representation of such dependencies, constraint class are introduced to the configuration model.

The constraint class is a logical implication that can be explained as *if…then* statements. If some classes or properties are evaluated positively at product configuration and they are connected to incoming Link, then all classes connected as outgoing Links must be set as logically implied. The Constraint Link may have a negative behaviour, and then classes connected to outgoing Links are rejected from the product variant.

The example provided in figure 3 displays class A, which has the structure of two part classes, P1 and P2.

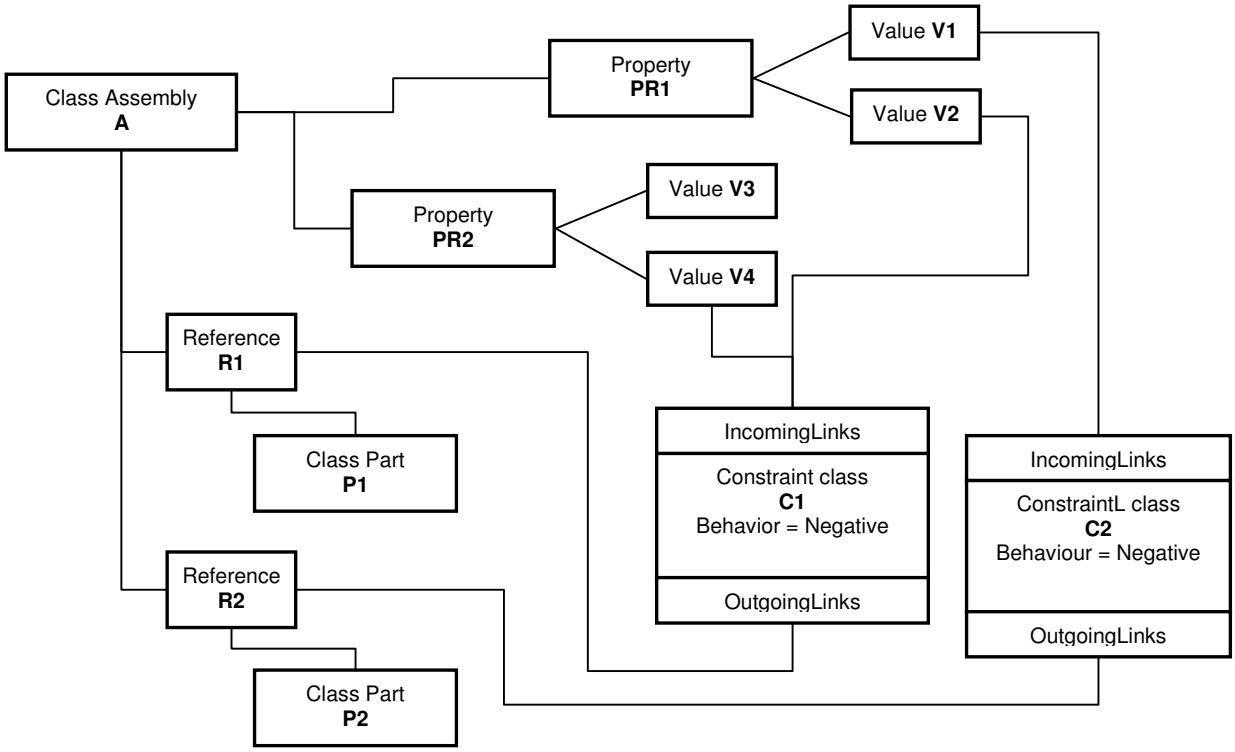

Fig 3. Usage of constraint classes

These classes are not linked directly to the assembly class A, but by the reference objects R1 and R2. Reference objects are necessary to store the quantity of referred part classes. Class A is linked to property classes PR1 and PR2, and each of them has several values. It follows that there are two questions for a user to answer in order to get a fully defined class A. A particular value has to be chosen for every property.

In this example, properties are used to control the structure of the main class. Constraint class C1 is created to eliminate P1 by disabling reference object R1 if the user selects values V4 or V2. Analogically, constraint C2 excludes P2 from the structure, if value V1 is selected.

Constraints can link various elements of the same class and classes that are not directly related. Usage of constraints allows one to define various exceptions that would be difficult to express by inheritance.

9. Conclusions

This paper provides an overview of currently existing solutions for the product configuration problem and emphasizes open issues. To address them, a formal representation of a configuration model, which is a net of interconnected classes related to product structure, is defined. The proposed configuration model allows easy modifications and addition of new variants.

References

Beuche, D.; Papajewski, H.; Schröder-Preikschaft, W. 2004. Variability management with feature models. In *Proceedings of Software Variability Management Workshop.* University of Groningen.

Dechter, R. 1990. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41: 273–312.

Gottlob, G.; Leone, N.; Scarcello, F. 2000. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124: 243–282.

Fleischanderl, G.; Friedrich, G.; Haselbock, A. *et al.* 1998. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems, Special Issue on Configuration*, 13(4): 59–68.

Felfernig, A.; Friedrich, G.; Jannach, D. *et al.* 2001. Distributed configuration as distributed dynamic constraint satisfaction. In *Proceedings of the 14th IEA/AIE.* Budapest, Hungary, 434–444.

Freuder, E. C.; Carchrae, T.; Beck, J. C. 2003. Satisfaction guaranteed. In *IJCA Workshop on Configuration, Eighteenth International Joint Conference on Artificial Intelligence.*

Hadzic, T.; Subbarayan, S.; Jensen, R. M. *et al.* 2004. Fast backtrack-free product configuration using a precompiled solution space representation. In *Proceedings of the International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems.*

Magro, D.; Torasso, P. 2001. Supporting product configuration in a virtual store. *LNAI*, 2175: 176–188.

McGuinness, D. L.; Wright, J. R. 1998. An industrial-strength description logic-based configurator platform. *IEEE Intelligent Systems.* July/August 69–77.

McGuiness, D. L.; Wright, J. R. 1998. Conceptual modelling for configuration: a description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, Special Issue: Configuration design*, 12(4): 333–344.

Soininen, T.; Gelle, E.; Niemel, I. 1999. A fixpoint definition of dynamic constraint satisfaction. In *Proc. of 5th International Conference on Principles and Practice of Constraint Programming - CP99*. Alexandria, USA, 419–433.

Subbarayan, S.; Jensen, R. M.; Hadzic, T. *et al.* 2004. Comparing two implementations of a complete and backtrack-free interactive configurator. In *CP 2004 Workshop on CSP Techniques with Immediate Application*.

Yokoo, M. 2001. *Distributed constraint satisfaction - foundations of cooperation in multi-agent systems*. Berlin: Springer.

Zhang, Wen-lei; Fan, Yu-shun; Yin, Chao-win. 2006. Approach of product configuration based on product family genealogy. In *Proc. of Shenyang Institute of Automation.* Shenyang, 12(11): 1741–1746.

GAMINIO KONFIGŪRAVIMO SISTEMOS, SKIRTOS GAMINIO DUOMENŲ VALDYMO SISTEMAI, ABSTRAKTI STRŪKTŪRA

L. Burneika

S a n t r a u k a

Gaminio konfigūratorius yra programinė sistema, pateikianti klausimus apie produktą su galimais pasirinkimų variantais. Šiame darbe pasiūlyta nauja konfigūratoriaus idėja ir pateiktas konfigūracijų aprašo modelis. Šis modelis buvo kuriamas taip, kad juo būtų galima aprašyti realius, dažnai tobulinamus ir sudėtingos sandaros gaminius. Modelyje informacija apie gaminį pateikiama kaip tarpusavyje sujungtų įvairių klasių tinklas. Modelio klasėse saugoma informacija apie gaminio junginius, komponentų ryšius ir loginius apribojimus. Siūlomas konfigūracijų aprašo modelis leidžia lanksčiai perteikti inžinerines žinias apie įvairių gaminių, tarp jų ir aviacijos reikmėms skirtų gaminių, variantus.

Reikšminiai žodžiai: gaminio konfigūravimo sistema, gaminio duomenų valdymo sistema, žinių vaizdavimo modeliai.