ISSN 1392 - 1207. MECHANIKA. 2008. Nr.3(71)

Product configurator for product data management system: design of internal structure

L. Burneika
Vilnius Gediminas Technical University, J. Basanavičiaus 28, 03224 Vilnius, Lithuania, E-mail: Linas@mail.lt

1. Introduction

Product configuration problem emerged as a research topic in the 1980s as the result of manufacturing companies going from mass-production to mass customization. Configurators not only provide realization of the mass-customization paradigm but also are among the most successful applications of artificial intelligence technology. In product configuration process a user makes adjustments to the product (an industrial machine, a car, a personal computer etc.) according his specific needs, using supporting software called the configurator. It calculates a specific product variant that fulfills provided requirements, technical and nontechnical constraints. Choices for each available component are usually modeled as variables over finite domains, and the knowledge about the valid product specifications is encoded as propositional constraints over these variables.

A tendency is observed that the economy of the 21st century will be based on highly specialized solution providers working according concepts of agile manufacturing [1]. Therefore the configuration of constantly evolving and changing products must be supported. 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 for the case.

2. Related works

There is a long history in 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 Problem (CSP) [2], or description logics [3].

Configuration tasks are more dynamic in nature and therefore a CSP representation, where all problem variables must be known from the beginning, is not appropriate in many application domains. In order to solve this, the dynamic constraint satisfaction [2] was developed, which is more suitable for representing and solving configuration tasks, because the set of problem variables may vary according to some activity constraints. In reference [4 - 6] a distributed dynamic CSP is defined and a modification of the asynchronous backtracking algorithm from [7] is applied for problem solving. The limitations of dynamic CSP representation becomes evident, when configuring large technical systems. To overcome this problem a generic CSP representation [8] has been proposed. There, new instances of problem variables can be created from meta-variables during problem solving.

In CSP approaches [9, 10], the configuration can exploit powerful constraint problem solvers for solving complex problems [11]. The alternative symbolic approach [12] has to split computations to an offline and an online phase. First they compile valid user assignments to efficient data structures, such as reduced ordered Binary Decision Diagrams [13]. 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 [13], but considering user-friendliness requirement for configurator, both approaches have difficulties to comprehend representation formalism. Although commercial configurators have a user interface layer for simplification, but it does not solve numerous problems arising during frequent production updates. Another problem is that the process of constructing configurable product requires additional skills and is very different from product design tasks.

The ideas for product configurator proposed in this paper relate to previous research projects such as [14, 15]. They are looking for possible ways of configuration problem decomposition in order to improve solving efficiency and apply methods of parallel computing. My approach for product configuration evolves from similar concepts – configurational knowledge representation is decomposed and closely related to product structure. However representation formalism is not symbolic, but object driven. The objects of model are connected mostly according product structure and have references to engineering data on Product Date Management (PDM) system. The task of this work is to express information about product configurations as a net of interconnected objects, further called the configuration model, and provide architectural details for further implementation as the software system.

The product configuration model defined in this paper is not ordinary sales configurator, but falls into engineer-to-order category. The configuration model is designed taking into account high structural complexity of real life products that are dynamic by nature and undergo continuous improvements. This is the reason why architecture of model includes close relations with PDM system.

The configuration model has dualism in the way product variants are expressed: by class inheritance and by constraint links. Another original feature of the model is an object driven approach, allowing to add new variants faster and easier than conventional configurators.

3. The configuration model - multilayer structure

The structure configuration model consists of two layers: the understandable for user abstract layers, and physical layer for the implementation. Abstract layer deals

with assembly, property and constraint abstract classes. They are closely related to the usage of product configurator. Physical layer provides information about the implementation of configuration model as software system, therefore its structure is different from abstract layer. It is described by Unified Modeling Language (UML). Every entity of abstract configuration model layer is represented by instances of C++ classes that further on this paper are called objects.

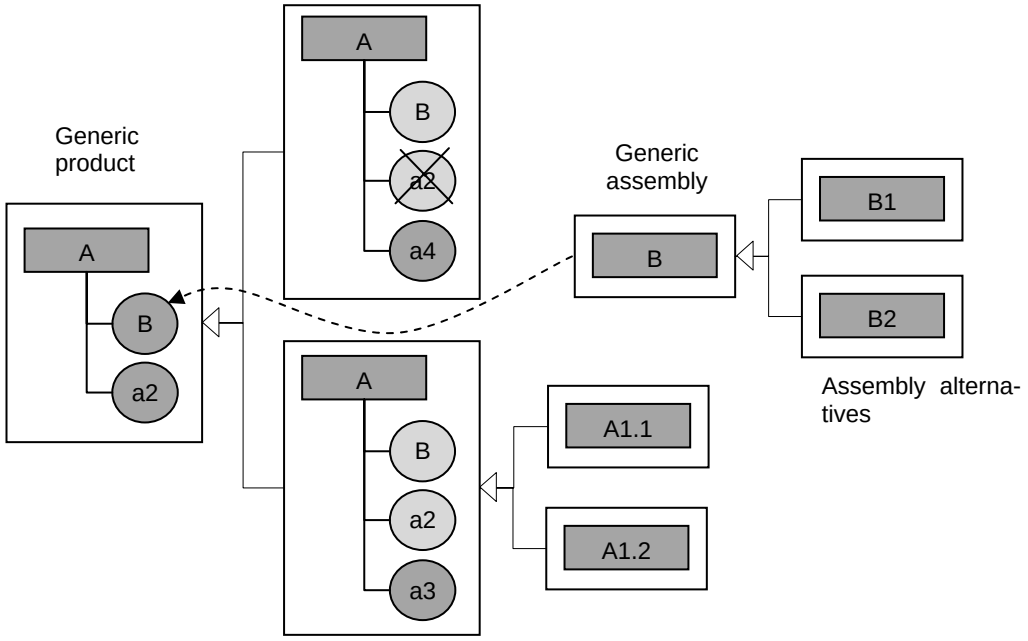

Fig. 1 Generic class inheritance example in configuration model

Product configuration model is constructed using concepts of object oriented modeling and classification of components, where the main entity is an abstract generic assembly class. Every abstract generic assembly class may contain references to other abstract classes and items from PDM system. An assembly class is an abstraction of real product assembly. In the Fig.1 class *A* defines a generic assembly, which always contains at least two components: another assembly class *B* and component *a2*. Other two classes are derived from base assembly class *A*, where each derived class introduces additional information. Class *A1* has extended assembly structure by a new component *a3*, while class *A2* overrides inherited component *a2* by *a4*. Abstract classes *A1.1* and *A1.2* are inherited from *A1* for addition of even more details into product structure, however for clarification this is not shown in figure. Every abstract class existing at a leave node of inheritance tree represents a particular configuration of a root class. In this example the abstract class *A* has three configurations. A generic 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 six configurations in total.

Experience shows, that in most cases new product variants are designed by finding existing, the most closely fulfilling customer requirements product, and modifying it. Therefore new design efforts are minimized by heavily reusing previous work. The principle of inheritance provides an easy way to add new variants to configuration model, but existing variants are conserved.

Choices for product configuration can be easily determined from the model. In the example inheritance tree of class *A* is a question itself with 3 possible choices for the user: class *A2*, *A1.1* and *A1.2*. The tree of class *B* provides two choices. During configuration process of this example product user has to answer two questions in order to get fully defined assembly of the class *A*.

Ability to define structure for a class and inheritance in the configuration model provide tools for compact description of multiple product variants. It is also easier to understand for engineer and has flexibility for making changes in configurations.

4. Physical layer - design of internal structure

4.1. Properties

In abstract layer of configuration model properties add some static information to generic classes. In physical layer there are three types of property classes: enumeration (*PropEnum*), range of values (*PropRange*) and arbitrary value (*PropArbitrary*). An object of enumeration contains links to a fixed list of values that are hold in objects of *ValueHolder* type. The *ValueHolder* object may have at least one value, however in some cases there can be more values of different types. A C++ class template *Value* lets to instantiate objects of simple data types, like integer, real or string. It is also possible to have some fixed additional values on *PropEnum* and each *ValueHolder* objects. In Fig.2 this is represented as on-to-many relations.

Property range itself is a class template, which allows creations of objects of primitive data types. This object holds information about upper and lower limits of the

range as well as increment. *PropRange* may have additional fixed values the same way as in enumeration object.

Very similar to property range is the property of arbitrary value. It is also a class template, so objects of different data types can be instantiated. *PropArbitrary* object may have additional fixed values.

For convenience objects of properties are linked by pointers to *PropGroup* objects. Any property objects of different type can be grouped together. *PropGroup* object is a self linked object, therefore a tree shaped structure can be created for the classification of other properties.

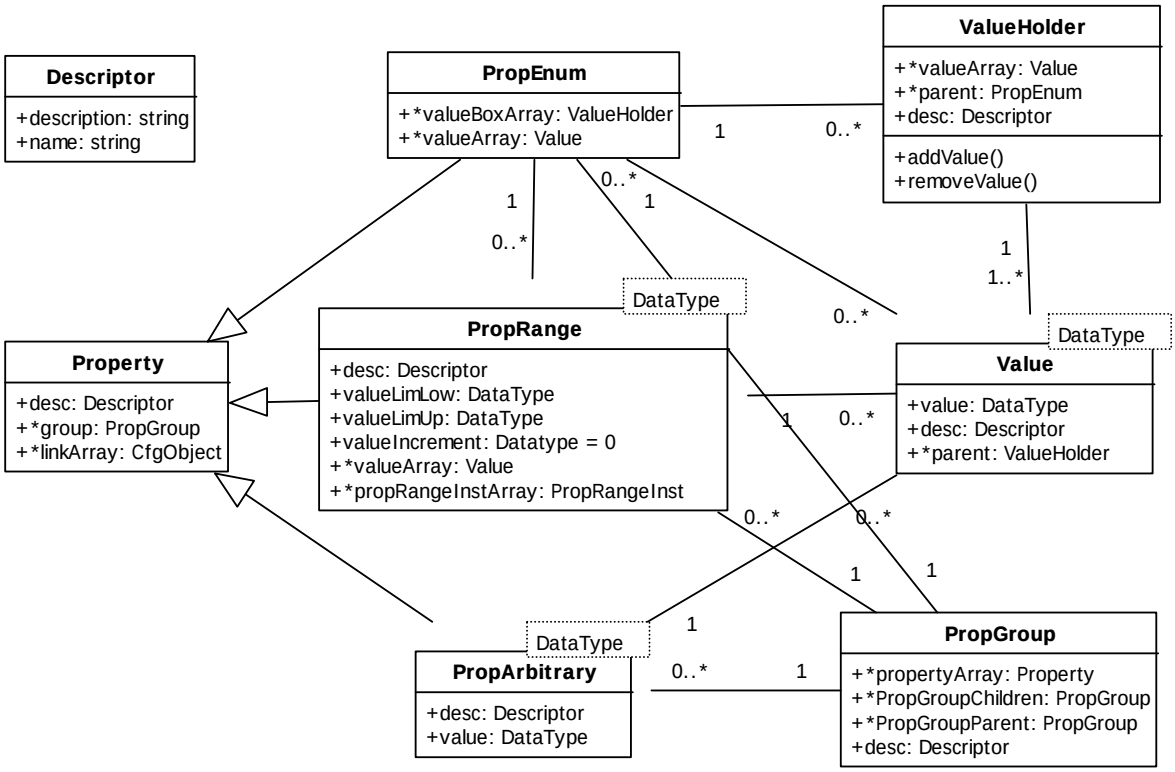

Fig. 2 Class diagram for properties

4.2. Main objects

The central class of physical model is *CfgObject*, which contains a common functionality for evaluating particular product variant, according user choices. Other classes are derived from *CfgObject*. A backbone of configuration model is constructed from *AssemblyItem* and *PartItem* classes that are derived from *AbstractItem* (Fig. 3). Every generic "assembly class" is an object instantiated from *AssemblyItem* class, shown in Fig. 3. "Generic derived assembly classes" are also separate objects of *AssemblyItem* type, but they are linked by pointers to each other. PartItem is a simple object, containing reference of an item taken from the PDM system.

Object of *AssemblyItem* may have multiple links to another *AssemblyItem* or *PartItem* objects. This is accomplished by using objects of *AbstractLink* type. There are two types of links. The simplest link *PdmItemLink* holds an identification of item in PDM system. This type of the link is useful for attaching documents from PDM system to objects of configurator. *ItemLink* object can hold one pointer to any object of *AbstractItem* type. *ItemLink* has information about quantity and units of measurement. Using objects of *AbstractItem* and *AbstractLink* a recursive tree of generalized parts and assemblies can be constructed.

A special type of object is *CommonFeature*. This object contains pointer references to *AbstractItem* objects sharing the same set of properties. *CommonFeature* objects may be derived from each other for user convenience. Similarly as in *AbstractItem*, information about object-to-object inheritance is represented by arrays of pointers.

Alternative components are referred to the components, which are completely replaceable from engineer's point of view. This is a very often situation in manufacturing industry. Object of *AlternativeItems* type creates a list of replaceable components by using the array of pointers to *AbstractItem* objects. The effect of *AlternativeItems* object can be defined by setting its scope. If the scope refers to particular *AssemblyItem* object, then component alternatives are local to this object.

Finally, object of class Product defines a space for particular family of designed and manufactured products. Every other object has reference to Product object. However object cannot belong to more than one Product objects simultaneously. This allows having different configuration models for different products, but parts of configuration model cannot be shared among products.

4.3. Constraint links

Some information about choices is represented by inheritance of abstract generic classes. But in reality more complex interdependencies between components usually exists. For the representation of such dependencies *ConstraintLink* class are introduced to configuration model.

The *ConstraintLink* object is a logical implication, that can be explained as If…Then statements. If some objects evaluated positively on product configuration are connected to *incommingLink*, then all objects connected as *outgoingLinks* must be set as logically implied. The *ConstraintLink* may have a negative behavior, then objects connected to *outgoingLinks* are rejected from product variant.

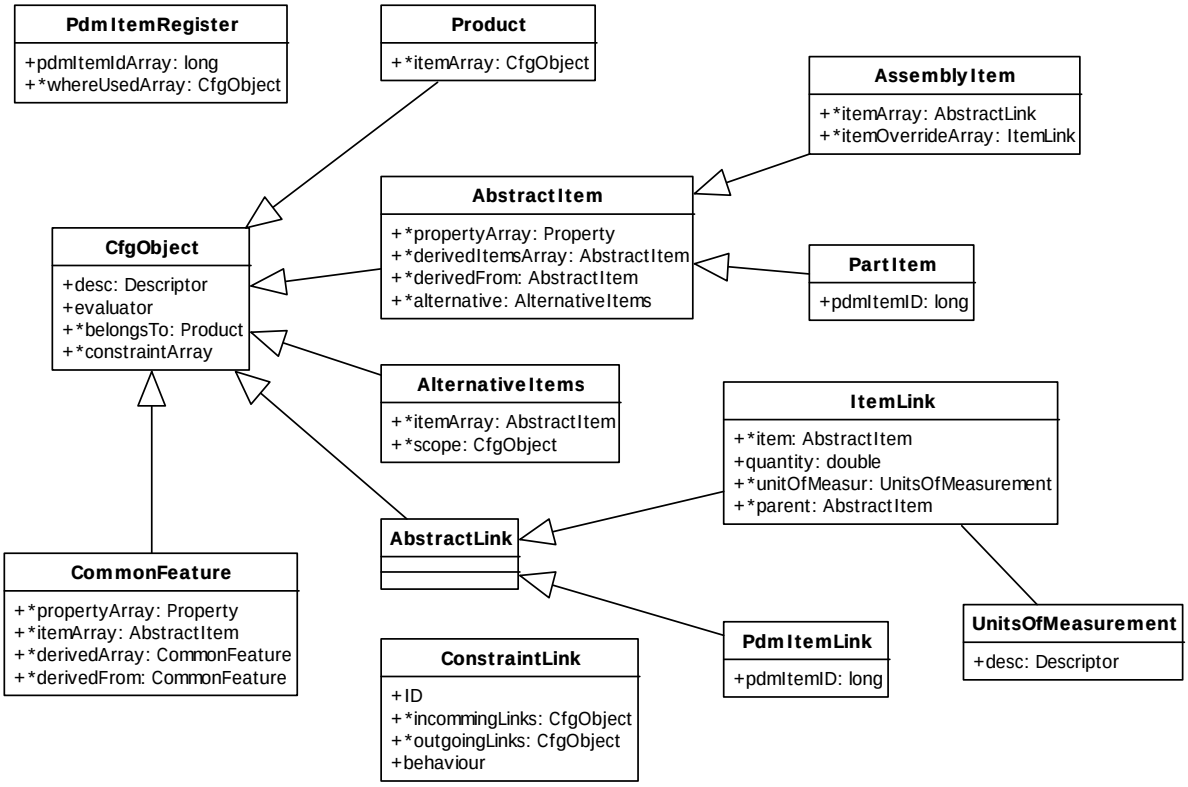

Fig. 3 Class diagram for the main classes of physical configuration model

5. Case study for configuration of laser micromachining device

As an illustration of application of product configurator, this section provides a small example based on laser cutting device. Lasers are a powerful tool for micromachining applications. A focused laser beam is concentrated onto a small target of a few microns in diameter. The laser-material interaction in this target area is controlled by laser parameters such as wavelength, pulse energy and pulse duration. All these parameters depend on material and cutting regimes. To achieve proper cutting under various conditions a different hardware is necessary. To provide a solution, manufacturing company offers multiple different laser models, each of them has many optional components, suitable for specific tasks.

The purpose of example in Fig. 4 is to show how variants of a product could be defined by principles of the configuration model. However only a small subset of device is analyzed, explaining several components and options. The main abstract assembly class “Picosecond mode locked Nd:YAG laser” contains everything, that is common to all other laser models of this type. There are Control panel, Casing and Power supply in the structure of the class. The components of this class belong to every model, regardless of other technical parameters. However this class already provides a possibility to choose power supply (single phase or three phases) and PC interface (RF232 or USB).

In order to implement various pulse energy outputs, different laser models are designed in classes PL1, PL2 and PL3, which are derived from the main class. The output parameters of laser depend on design of Optical block. A new class representing this component is created and added into structure of PL1 model. The standard laser wavelength is 1064 nm, however other wavelengths are also available. For this reason new classes are created deriving the Optical block class. The new classes have different structure and components, implementing necessary functions. The structure of standard optical block is shown in more detail on Fig. 4. Alternative optical blocks, shown as Option1 and Option2, may have different design and structure.

All classes are enclosed by Picosecond laser class, forming the complete configuration model for the product. Usage of the configuration model begins from the root class – the user has to select one laser model from three alternatives. Once it is done - PC interface and power supply must be determined. If PL1 model was chosen initially, then optical block must be selected as well.

From the physical layer point of view, every solid line rectangle in the diagram represents AssemblyItem object. The structure of AssemblyObject is enclosed by dashed lines. Every thin line representing structural link is the ItemLink objects. In the Standard optical block class usage of ItemLink objects are shown explicitly. Links between AssemblyItem objects represent inheritance – it is shown as white arrows pointing to the parent object, from which derivatives are created.

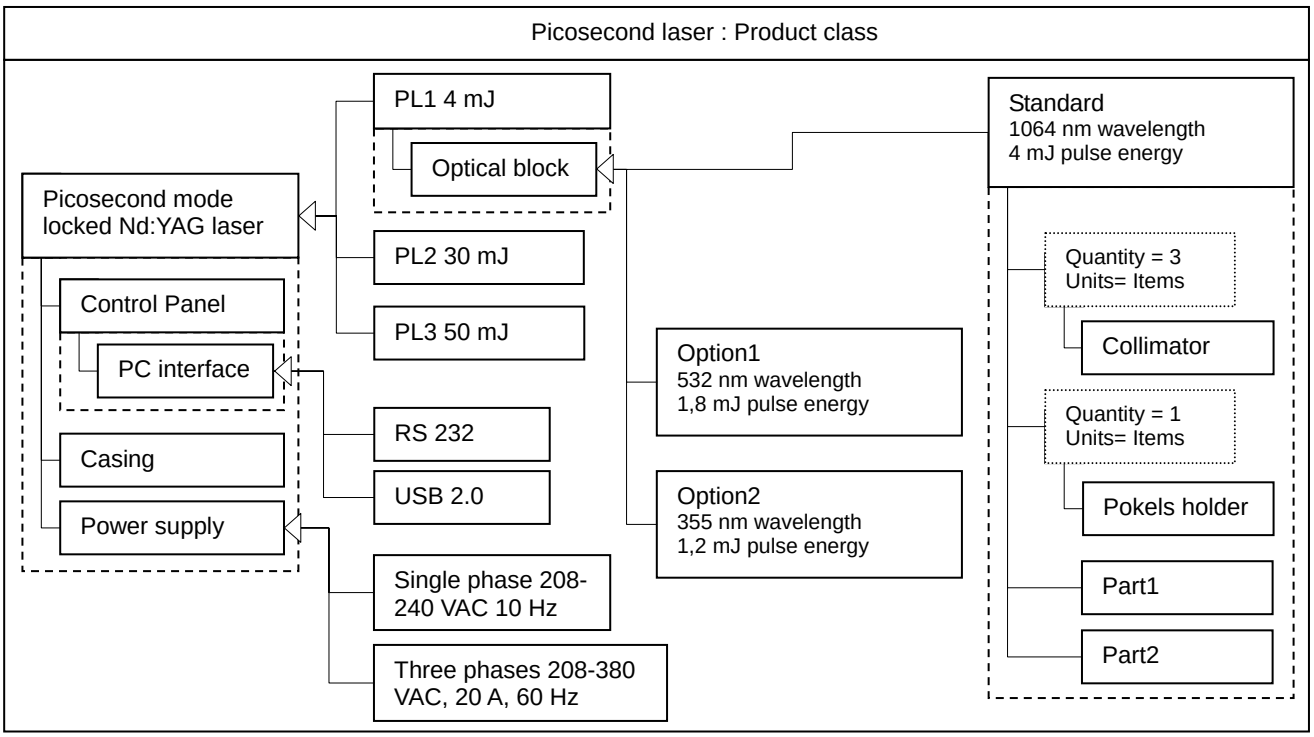

Fig. 4 Example of configuration model for micromachining device

6. Conclusions

1. In this paper a formal representation of configuration model is created and defined as net of interconnected objects, closely related to product structure. Configuration model is designed to allow representation of multiple product variants logically and compactly. New variants can be added by inserting classes or deriving existing ones. Usage of product configurator system, based on configuration model would streamline engineering work and preparation of product variant documentation for manufacturing.

2. The product configuration model includes description of object types, attributes and architecture schemes, intended for further implementation of configuration model as a software system.

References

1. **Anužienė, L., Bargelis, A.** Decision support system framework for agile manufacturing of mechanical products. -Mechanika. -Kaunas: Technologija, 2007, Nr.3(65), p.51-57.
2. **Soininen, T., Gelle, E., Niemel, I.** A Fixpoint definition of dynamic constraint satisfaction.-In 5th International Conference on Principles and Practice of Constraint Programming - CP99. -Alexandria, USA, 1999, p.419-433.
3. **McGuinness, D.L., Wright, J.R.** Conceptual modeling for configuration: a description logic-based approach. artificial intelligence for engineering design.-Analysis and Manufacturing, Special Issue: Configuration Design, 1998, 12(4), p.333-344.
4. **Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.** Distributed configuration as distributed dynamic constraint satisfaction. -In Proceedings of the 14th IEA/AIE, Budapest, Hungary, 2001, p.434-444.
5. **Yokoo, M.** Distributed constraint satisfaction - foundations of cooperation in multi-agent systems.-Springer, Berlin, Germany, 2001, p.47-54.
6. **Bargelis, A., Česnulevičius, A., Stasiškis, A., Šačkus, A.** Intelligent manufacturing engineering based on multi agent tools. -Mechanika. -Kaunas: Technologija, 2003, Nr.1(39), p.40-48.
7. **Dechter, R.** Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition.-Artificial Intelligence, 1990, 41(3), p.273-312.
8. **Fleischanderl, G., Friedrich, G., Haselbock, A., Schreiner, H., Stumptner, M.** Configuring large systems using generative constraint satisfaction.-IEEE Intelligent Systems, Special Issue on Configuration, 1998, 15(17), 13(4), p.59-68.
9. **Freuder, E.C., Carchrae, T., Beck, J.C.** Satisfaction guaranteed. IJCA workshop on configuration. -Eighteenth Int. Joint Conf. on Artificial Intelligence, 2003, 15(17), p.646-651.
10. **Gottlob, G., Leone, N., Scarcello, F.** A comparison of structural CSP decomposition methods. Artificial Intelligence, 2000, 124(2), p.243–282.
11. **Beuche D., Papajewski, H., Schröder-Preikschaft, W.** Variability management with feature models. proceedings of software variability management workshop.-University of Groningen, 2004, 53(3), p.333-352.
12. **Hadzic, T., Subbarayan, S., Jensen, R.M., Andersen, H.R., Møller, J., Hulgaard, H.** Fast backtrack-free product configuration using a precompiled solution space representation.-Proc. of the Int. Conf. on Economic, Technical and Organizational aspects of Product Configuration Systems, 2004, p.28-29.
13. **Subbarayan, S., Jensen, R.M., Hadzic, T., Andersen, H.R., Hulgaard, H., Møller, J.** Comparing two im-

plementations of a complete and backtrack-free interactive configurator.-CP 2004 Workshop on CSP Techniques with Immediate Application, 2004, p.97-111.
14. **Wen-lei Zhang, Yu-shun Fan, Chao-win Yin.** Approach of product configuration based on product family genealogy.-Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, 2006, 12(11), p.1741-1746.
15. **Magro, D., Torasso, P.** Supporting product configuration in a virtual store.-Lecture Notes In Computer Science, Proc. of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence, 2001, v.2175, p.176-188.

L. Burneika

GAMINIO KONFIGŪRATORIUS SKIRTAS GAMINIO DUOMENŲ VALDYMO SISTEMAI: VIDINĖ SANDARA

R e z i u m ė

Šiame darbe pasiūlyta nauja gaminio konfigūratoriaus idėja ir pateiktas konfigūracijų aprašo modelis. Modelis buvo kuriamas taip, kad juo būtų galima aprašyti realius sudėtingos sandaros gaminius, kurie dažnai tobulinami. Modelyje informacija apie gaminį pateikiama kaip tarpusavyje sujungtų įvairių tipų objektų tinklas. Objektuose saugoma informacija apie gaminio junginius, komponentų ryšius ir loginius apribojimus. Kai kurių tipų objektai jungiami su gaminio duomenų valdymo sistemoje (PDM) saugomais dokumentais. Siūlomas konfigūracijų aprašo modelis leidžia lanksčiai perteikti inžinerines žinias apie gaminio variantus.

L. Burneika

PRODUCT CONFIGURATOR FOR PRODUCT DATA MANAGEMENT SYSTEM: DESIGN OF INTERNAL STRUCTURE

S u m m a r y

This work proposes a new idea of product configurator and defines a configuration model. The configuration model is designed taking into account high structural complexity of the real life products that are dynamic by nature and undergo continuous improvements. Information about product in the model is expressed as a structured net of interconnected objects of different types. Objects hold information about product assemblies, structural links and logical constraints. Some objects have references to documents or items on Product Data Management system. The proposed configuration model allows flexible representation of knowledge about product configurations.

L. Burneika

КОНФИГУРАТОР ПРОДУКТА ДЛЯ СИСТЕМЫ УПРАВЛЕНИЯ ДАННЫХ ПРОДУКТА: ПРОЕКТ ВНУТРЕННЕЙ СТРУКТУРЫ

Р е з ю м е

Работа предлагает новую идею конфигуратора продукта и определяет модель конфигурации. Модель конфигурации разработана принимая во внимание высокую структурную сложность реальных продуктов, которые являются динамическими по своей природе и подвергаются непрерывным усовершенствованиям. Информация о продукте выражена как структурированная сеть связанных объектов различных типов. Объекты содержат информацию о сборках продукта, структурных связях и логических ограничениях. Некоторые объекты имеют ссылки на документы в системе управления данных продукта. Предложенная модель конфигурации позволяет гибкое представление знании о конфигурациях продукта.

Received January 04, 2008

DOI: 10.5755/j02.mech.15082