From CSP to configuration problems

Mathieu Veron
Access Productique, Le Stratège
Bât B2, Rue Ampère, BP 555,
31674 Labège Cedex, France.
mv@access-pro.fr

Hélène Fargier
Institut de Recherche en
Informatique de Toulouse 118
route de Narbonne, 31062
Toulouse Cedex 4, France.
fargier@irit.fr

Michel Aldanondo
Centre de Genie Industriel, Ecole
des Mines d'Albi Carmaux,
Campus Jarlard Route de Teillet,
81013 Albi CT, Cedex 09, France.
Michel.Aldanondo@enstimac.fr

Abstract

To increase market shares, industry needs to provide customized products, at a low price and low delivery time. But establishing a valid configuration is a complex, time-consuming and costly task. There is a need for software tools to help people to model their products, and allow them to compute a valid configuration.
The CSP framework seems to be a valuable candidate to express models for configurable product and to solve the configuration problem. In this paper, we will show, on the basis of our experience in the field, that we need a richer model to capture the specificity of a configurable product and other functionalities to easily handle the resolution process. We will review some previous work fulfilling some of these functionalities and finally present an approach able to handle the whole problem.

Introduction

Configuring a product is to choose a feasible instance of this product among all its variations. Hence configuration is a search problem over a search space defined by a model of a configurable product. This model is composed of the description of all the attributes characterizing it, the allowed values for these attributes and the constraints expressing incompatible values (Mittal & Frayman 1989).
According to this definition of a model, the Configuration Problem can be mapped into a constraint problem. Whereas the configuration software (configurators) were implemented like expert systems (R11/XCON, McDermot 1982), the Constraint Satisfaction Problem (CSP) framework is now preferred due to its knowledge representation which is declarative and context free (for a further discussion about pro and cons of both approaches see Gelle and Weigel 1996).
Our experience in the field of configurator and configurable product modeling, acquired after near ten years of activity in the configurator industry, points out that the CSP framework is not able to the specificity of the configuration problem capture entirely and easily, hence the need for a more complex model to represent a configurable product and the relevant algorithms.
After reviewing, in a first section, a list of further requirements. We will analyze how previous work in the field handle these needs and the solutions they provide to manage them, with a particular focus on constraint based approach. Finally we will provide some ideas for the problem definition and its resolution. The conclusion will introduce some further work on this model.

Requirements to a configuration problem

Basic Configuration Problem

A configurable product is classically defined by a set of attributes (or components) which possible values belong to a finite set, and a set of feasibility constraints over these attributes which specify their compatible combinations of values. The problem is to find a feasible product (i.e. to choose a value for each attribute) that satisfies not only the feasibility constraints but also some user requirements. In a first basic approach one could consider that each of these requirements concerns one attribute. In this case, the Constraint Satisfaction Problem (CSP) (Mackworth 1992) offers a suitable framework. A CSP is indeed described by a triplet = {X, D, C}, where X is a set of variables, D a set of finite domains (one for each variable) and C a set of constraints. A constraint $c \in C$ is defined by a set $S(c)=\{i,j,...m\}$ variables and R(c) a subset of $Di \times Dj \times ... \times Dm$ expressing the combinations of instantiations of these variables that satisfy the constraint. A constraint c is said to be valid with respect to a partial instantiation s, if the projection of s over S(c) is included in R(c). A solution to a CSP problem is an instantiation of all variables such that all constraints are valid.
The mapping is then obvious: the variables are the attributes of the product, and the constraints encode both the requirements (by means of unary constraints) and the feasibility constraints.

Need to manage the states of the objects

In a classical CSP, a solution is a consistent instantiation of all the variables. But in a configuration problem, all the variables do not need to be valuated in order to consider the configuration completed. On one hand, the valuation of some variables is optional (the optional variables can be

left uninstantiated at the end of the configuration process). Hence, the set of variables X has to be partitioned into two subsets Xm and Xo (the mandatory variables and the optional variables) and an instantiation d of a set of variables Y such that Y included Xm is a solution of the configuration problem if and only if it satisfies all the constraints pertaining to Y.
On the other hand, some variables are exclusive, in the way that the valuation of the first one forbids the valuation of the second one and reciprocally: they cannot participate in a solution simultaneously. For instance, the user cannot choose simultaneously a CD drive and a DVD drive when configuring a PC. Moreover, the possibility to provide a value to a variable can depend on the value taken by another one: as soon as the type of the disk driver is SCSI, the user has to choose a SCSI card - but he cannot configure such a card if he has chosen an IDE driver.

Hence, we need to handle the notion of state of a variable (active or not, if active optional or mandatory) and to encode the previous activity conditions over the states. A first solution could be to add a dummy value within the domain of each variable (let's name it θ) and to modify the set of constraints in order to capture the activity conditions: θ is the only valid value for y if the activation condition of y is not satisfied. Although this kind of approach allows the use of the classical CSP framework, it requires to merge feasibility constraints and activity conditions. Handling the states of the variables more explicitly not only overcomes this drawback but also allows the expression of other possible states (e.g. optional, mandatory, etc).

Need to manage the Structural decomposition

Structural decomposition is a strong issue for the configuration problem. The technical analysis of a configurable product clearly shows the interest of a decomposition in sub-components that can involve their own internal constraints. For products made from the assembly of sub-parts, this decomposition is mainly the bill of materials. But, in more complex configurations, a sub-component can be itself a configurable product. Hence we distinguish the notion of elementary configurable product: an elementary configurable product is a self-containing entity defined as previously in terms of attributes, domains and internal constraints. In this context, a configurable product can be:

- either a single elementary configurable product,
- or a collection of :
  - standard components (not configurable product as raw material or bought components)
  - and/or configurable products (elementary or not), the attributes of which can be related by a set of transversal constraints, i.e. constraint on the value of any attribute.

Under this assumption, we need to slightly modify the definition of a complete configuration in the following way: an elementary configurable product is completed if and only if a consistent instantiation for all its active and mandatory variables exists. A configurable product is then completed if and only if all its mandatory sub-components are completed.

In order to be able to easily reuse sub-components inside another configurable product, we have identified a condition of self-sufficiency for configurable products. This condition states that no outside knowledge is necessary, all the references are internal. In our model, this is translated by a property on the constraints:

- the transversal constraints only involve attributes of configurable products (elementary or not) included inside the configurable product that need to be reused;
- the state conditions only involve state variables of attributes and configurable products (elementary or not) included inside the configurable product that need to be reused.

We are now able to reuse any sub-component into other configurable products. For example a car manufacturer, who sells cars and vans, can isolate a sub-product "seat" in each configurable product. The sub-product "seat" is modeled only once and reused in both configurable products (Car and Van).
Another benefit of such a decomposition deals with the expression of activity conditions: a unique rule can describe the fact that the same condition controls the existence of all the variables of a sub-product.

Need to make the distinction between functions and standard components

Configuration problems can be classified according to their complexity (Gartner Group 1997). From the lowest level of complexity to the highest, we most often find:

- Pick-to-order (PTO) problems, standard components are just picked from a catalog, with very few compatibility constraints on the possible configurations.
- Assemble-to-order (ATO) problems, standard components must be chosen according to compatibility constraints plus some constraints that restrict the possible assemblies. The classical example of ATO is personal computer configuration.
- Build-to-order (BTO) problems, components are not required to be standard, but can be tailored. In this case, the product can be structured into sub-products and recursively.

In Build-to-order problems, the physical components of the decomposition are not identified, the only specification the user can give is in term of functions. But even in problem involving only standard components, the user does not always hold the required technical background to choose

standards components and can find interest in expressing functional needs. For instance, a PC buyer does not always know the difference between one hard drive and another, but he knows that he wants a fast hard drive with a large capacity. The description of functional needs requires the use of both discrete and continuous variables.
Moreover, in any case after configuring functions, a matching from functions to components or, in the case of build-to-order, from functions to manufacturing orders has to be computed.

Need to manage the interactivity

The human user has a particular place in the configuration process. It is obvious for configurators dedicated to the selling process, but we have encountered many cases in the back-office where the problem and the optimization criteria could not been completely specified, and had to be left to human appreciation.
CSP algorithms are designed for batch processing, i.e. looking blindly for a solution. But, in a configuration process, the choice of the values is due to the user, interactively. The main goal to be achieved by a configurator is to guarantee at each step (i.e. after each user choice) that the partially specified product is feasible. In the CSP framework this could be expressed by enforcing global consistency after each user choice.
Because global consistency is a very costly operation, local consistency is often the only level of consistency that can be achieved in real-time. But in this case, a mechanism that warns the user of the detection of inconsistency needs to be provided; for example a "backtrack-point" could be identified to help the user to restore consistency.
More generally, providing help and explanations are key issues to configuration and Interactive CSP.
Moreover, our experience shows that some internal hidden attributes are often defined by the experts when modeling the configurable product. Such variables are not under the user control (i.e. cannot be valued interactively) but correspond to technical parameters or intermediate calculations and are used within functional constraints.

Parallel with previous research

Previous research on configurations has addressed one or more of those requirements. In the following we will review some previous work with an emphasis on constraint-based configurators.

Mittal and Frayman (Mittal and Frayman 1989) present a definition of the configuration problem. They work on personal computer configuration and their definition is oriented toward assemble-to-order. This seminal work clearly outlines the need for a functional definition of configurable products and for mapping mechanisms that find out the component decomposition. Nevertheless, it only deals with the assembly of a predefined set of components and so fails to support tailored products (i.e. build-to-order problems).

Esther Gelle and Rainer Weigel (Gelle and Weigel 1996) claim that the spectrum of configuration problems is wider than the assembly of predefined components. That's why they propose an incremental model that simultaneously handles discrete and continuous variables. Hence this model allows the expression of functional descriptions as well as component assemblies.
Gelle and Weigel actually propose to enhance Incremental CSP (Mittal and Falkeiner 1990) with an algorithm able to ensure global consistency over continuous domains after each user choice.
The idea behind ICSP for configuration tasks is that the solution space is often so huge and the interactions between variables so complex, that the whole problem cannot be handled entirely. That's why the problem is restricted to "active" variables. The activity of a variable is derived from special constraints called "Activity Constraints (AC)". The traditional "Compatibility Constraints (CC)" are evaluated once all the variables involved in the constraint are active.
The activity constraints are also used to represent configurable products. In Gelle and Wiegel's model, each object (i.e. configurable product) has a type and attributes, but the list of attributes is not predefined but is dynamically conditioned by the selected type. Let's take an example: the 'engine' object takes is type over two kinds of engines ('Gasoline' or 'Diesel'), which are described by two activity constraints:

AC1 : Engine=Gasoline → Engine.G1 $\in$ g1, Engine.G2 $\in$ g2, Engine.G3 $\in$ g3

AC2 : Engine=Diesel → Engine.D1 $\in$ d1, Engine.D2 $\in$ d2

Where g1,g2,g3 and d1,d2 are the respective domains of the attributes G1,G2,G3,D1,D2.

So if the type of engine is Gasoline then engine is described by three attributes: G1, G2 and G3; but if the type chosen were Diesel engine would be described by two attributes: D1 and D2.

To handle real assembly-to-order problems, attributes are considered as "connectors" and the domain of each attribute is the list of components that can be connected to that connector.

The activity state over object responds to our object state need, but it is not expressive enough due to its Boolean nature. Although the mechanism of Activity Constraint provides a uniform way to model configurable products and to manage an activity state for each object, this paradigm fails to isolate self-sufficient products (attributes and internal constraints) for reusability purpose. Indeed all the activity constraints are expressed at the same level and no structuration tool is provided that could help gathering all the constraints relevant for a particular sub-product.

These drawbacks are avoided by the Composite Constraint Satisfaction Problem (CCSP) framework proposed by (Freuder and Sabin 1996) to handle the Configuration Problem. Informally, CCSPs are CSPs where domains can contain sub-problems : when a variable is instantiated with a sub-problem, it is replaced by this CSP. More formally, given a CSP P=(V,Dv,Cv), if $Xi \in V$ is given the value P'=(Xv',Dv',Cv') then P becomes P''=($V \cup V' / \{Xi\}$, $Dv \cup Dv' / \{Dxi\}$, $Cv \cup Cv' \cup C\{v',v\} / C\{Xi\}$).
The main configurable product is thus represented by an initial CCSP, composed of some variables whose domains are sub-CCSPs : this sub-problem represents the direct sub-component of the main product. This defines a hierarchy which reflects the structural decomposition. The leaves of the tree correspond to our elementary configurable products.
Hence this model explicitly shows out the structural decomposition and the replacement mechanism keeps the problem as simple as possible. But it lacks mechanisms to handle full interactions with a human user. Indeed, suppose that the user had previously made a choice over a variable, thus selecting a sub-problem and that later, he changes his mind, he cannot modify his previous choice since the variable is no longer available due to the replacement mechanism. This could be overcome by maintaining all the variables within the problem, but the CCSP model would suffer an increase in complexity.

We can find in previous work some answers to the needs identified in the first section, but none manages all of them. Hence we will expose a (partial) solution in terms of data structure and management levels able to handle all these requirements.

How to handle these requirements

According to our analysis in Section II, one of the main requirements for a configurator is the ability to handle the structural decomposition of configurable products. That's why we propose to model a configurable product by a tree with :

- internal nodes representing the sub-configurable products (the root being the main one)
- leaves corresponding to variables of either an elementary configurable product or standard products.

It appears in Section II.2 that we need to manage the state of the components of the tree, that can be active or not (i.e. accessible to the user or not). Moreover, an active component can be (i) optional or mandatory and (ii) completed by the user or not.

Related to these two needs, we have identified a condition of self-sufficiency of configurable product (see section II.3). In order to guarantee this condition, we have defined a configurable product as :

- a collection of sub-components,
- a set of constraints such that each variable involved in these constraints appears in a sub-problem,
- one state variable for each sub-component,
- a set of state conditions over any state variable appearing in this product or in a sub-product.

For the sake of clarity, we will assume in the following that selecting a standard component is modeled by an attribute whose domain is the list of the codes of the standard components. A trivial matching mechanism is responsible for listing the selected one into the final bill of materials decomposition.

More formally:

Definition: a configurable product (CP for short) is a quartet { Lsc , Cv, S, Cs} where:

- Lsc is the list of the sub-components of CP. It can be partitioned into two subsets Lcp and Lecp respectively denoting a set of non elementary CPs and a set of elementary CPs. The set of included variables of a configurable product cp is denoted riv(cp) and recursively defined by :
  - riv(cp) is the set of variables of cp if cp is an elementary configurable product,
  - $riv(cp) = \cup_{cp' \in Lcp} riv(cp')$ if cp is a non elementary CP.
- C is a set of constraints over the set riv(cp). These constraints encode the feasible, (or unfeasible) combinations of values for the attributes.
- S is a set of state variables, one for each sub-component.
- Cs is a set of state conditions over the set of included state variables of the products or of their sub-components and recursively.

The idea underlying this definition is the use of a two level configuration engine. A first level is responsible for the management of the tree structure and for the control of the state variables, whereas the second level involves the CSP mechanisms to maintain the domains of the configuration variables. The user is responsible for adding/retracting unary constraints on the active variables in order to express his choices, a constraint propagation mechanism on the second level CSP will erase the inconsistent values in the other domain.

First level : object management

The main goal of this level is to maintain a state for each object of the tree structure. The value of a state variable depends on:

- (i) the value of the state variable that represents the upper product in the hierarchy: a sub-component cannot be active if is container is not active;

- (ii) the state of its sub-components: a configurable product is completed if and only if all its sub-components are completed;
- (iii) on user actions: a variable is completed whenever it has been restricted to a singleton by the user,
- (iv) on state conditions (Cs) inserted in the model to enforce a particular state.

Let us now explain what kind of state conditions we handle, and how they are managed. In Section II.2, we explain how conditions can describe that an object is optional or not and when an object is active or not.

Hence, a state variable can take four different values : inactive (0), optional (1), required (2), completed (3). A positive value thus means that the variable is active.

State conditions are encoded by means of constraints that can involve both state variables and configuration variables. We distinguish two types of state conditions:

- exclusion conditions : an exclusion condition between objects states that only one of the objects can be active at any time, e.g. : an exclusion between A and B means that state(A)>0 and state(B)>0 is not a valid combination;
- requirement conditions : the validity of the condition (involving state variables and/or attribute variables) implies that the object must be completed. e.g. : State(A) >1 implies State(B) >1, or package='Deluxe' implies State(extra_warranty)>1 .

Thus we define a constraint problem made of a set of configuration and state variables (each one taking its value in a finite domain {0,1,2,3}) and a set of constraints. In order to ensure a coherent configuration process, the states need to be consistent with each other; this implies maintaining global consistency over the CSP.

Here is an example of such a constraint problem on the basis of the previous example taken : let there be four variables, cd_drive ∈ {24x, 36x, 40x}, dvd_drive ∈ {2x, 5x}, option_package ∈ {none, std, deluxe} and extra_warranty ∈ {1year, 2years, 3years}.
To express that cd_drive and dvd_drive are exclusive and that the deluxe option package implies an extra warranty, we will define the following constraints :

state(cd_drive)	state(dvd_drive)
>1	=0
=0	>1

and

option_package	state(extra_warranty)
deluxe	• 2
none	• 1
std	• 1

Second Level : Constraint management on value

In order to deal with the basic requirements of a configuration problem, i.e. finding a consistent instantiation of a CSP representing a configurable product, on a second level, we manage a constraint problem gathering all the active variables and the constraints pertaining to active elementary configurable products. This constraint problem is interactively modified by the user who expresses restrictions or relaxations over the domains of the variables. Restrictions and relaxations are handled throughout unary constraints. Because the user is free to relax a previous restriction, we have chosen a dynamic CSP representation of the problem (Detcher and Detcher 1988) .
For us to guide the user toward a feasible product, the problem has to be kept consistent with the user choices. But the constraint problem can be significantly **large** (numerous variables and constraints), hence global consistency is often too costly to enforce, so we only enforce a level of local consistency. We have explored some of these levels and finally the good old arc-consistency (Mackworth 1977, Mohr and Henderson 1986, Bessière and Cordier 1993, Debruyne 1996 for the adaptation to dynamic CSP) has proven to be a valuable candidate. Our first experiments have shown little interest for stronger levels of consistency, except for enforcing singleton arc-consistency (Debruyne and Bessière 1997) on the domain of the variables picked by the user, before letting him modify it.

Interaction between the two levels

The two previously described levels are tightly coupled; interaction between the two levels occurs through (i) the shared variables, i.e. the variables that appear in the CSPs of both problems; (ii) the change of activity state of an object. Indeed, in order to keep the CSP as simple as possible, whenever a configurable product becomes active, the linked constraints are added, and are removed when it becomes inactive.

Conclusion

Configuration is becoming an important issue for companies' competitiveness. Previous research shows that the CSP framework is of interest when addressing the configuration problem, but as shown in section II this research does not succeed in taking into account all the specific requirements of configuration for industry. In terms of a solution we have provided a two-level approach to handle some of these requirements and to solve the configuration problem interactively. This approach has been implemented within the Caméléon software suite, an interactive selling system from Access Productique.
As previously mentioned, handling interactivity is a key issue in the configuration problem. Indeed, we need to provide easy to use software tools to people without

product technical knowledge in order to let them achieve a complex configuration. The extreme case would be a web-based configurator for very complex products. Hence we need to work on providing contextual help to the user. This help could either be on the first level (activity of sub-components) or on the second level (explanations about discarded values). We are currently exploring an approach based on the arc-consistency algorithm providing supports.

Acknowledgments

This work was granted by Access Productique (S.A.).

References

Bessière and Cordier 1993. Arc Consistency and Arc Consistency Again, In *Proceedings of the Eleventh National Conference on Artificial Intelligence,* 108-113. AAAI Press.

Detcher and Detcher 1988. Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 37-42, St Paul, Minnesota. AAAI Press

Debruyne 1996, Arc-Consistency in Dynamic CSPs Is No More Prohibitive. In Proceedings of the eighth International Conference on Tools With Artificial Intelligence, 299-306. Toulouse, France

Gartner Group 1997. Sales Configurators : Configuring Sales Success. *The report on Supply Chain Management,* October 1997.

Gelle,E. and Weigel, R. 1996. Interactive Configuration Using Constraint Satisfaction Techniques, Technical Report FS-96-03, Workshop on configuration, 37-44. AAAI Press.

Mackworth, A. 1977. Consistency in Networks of Relation. *Artificial Intelligence* 8:99-118.

Mackworth, A. 1992. Constraint Satisfaction. In Shapiro, S. (Ed.) Encyclopædia of Artificial Intelligence, 285-293. Wiley, NY.

McDermot 1982. R1 : A Rule-based Configurer of Computer System, *Artificial Intelligence*, 19(1):39-88.

Mittal, S. and Frayman 1989. Towards a generic model of configuration tasks, In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,1*395 – 1401.

Mittal, S. and Falkeiner 1990. Dynamic Constraint Satisfaction Problems. . In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 25-32. AAAI Press

Mohr and Henderson 1986. Arc And Path Consistency Revisited, *Artificial Intelligence* 28:225-233, 1986.

Sabin and Freuder, E. 1996. Configuration as Composite Constraint Satisfaction, Technical Report FS-96-03, Workshop on configuration. 28-36. AAAI Press.