

# How to Squeeze a Configurator into a Handheld Device

Homero M. Schneider, Marcos F. Espindola and Yuzo Iano

**Abstract** In this paper, it is presented the implementation of a solar-powered pumping system configurator based on a new approach for the modelling of the configuration process of product families. An important outcome of this approach is that if a few modelling conditions are satisfied, deriving product family members becomes a direct (backtrack-free) process. Consequently, a configuration problem-solving process that typically would require a high-performance computer can be squeezed into handheld devices as a standalone program.

**Keywords** Product family · Product configurator · Backtrack-free search · Solar-powered pumping system

## 1 Introduction

Solar-powered pumping systems (SPPS) have long become a viable economic option for rural areas and the widespread of this technological solution will have a great socioeconomic impact in many countries. Nevertheless, this is still a largely unexplored market segment which will require the increasing involvement of the private sector [1]. For a company interested in this market, sending trained technicians to distant sites in the countryside to visit potential customers is a costly

---

H. M. Schneider (✉) · M. F. Espindola  
Centre of Information Technology Renato Archer (CTI), Campinas, SP, Brazil  
e-mail: homero.schneider@cti.gov.br

M. F. Espindola  
e-mail: marcos.espindola@cti.gov.br

Y. Iano  
Electrical and Computer Engineering Faculty, State University of Campinas (Unicamp),  
Campinas, SP, Brazil  
e-mail: yuzo@decom.fee.unicamp.br

approach, not to mention the shortage of trained technicians in SPPS. A better approach is to provide an established sales force with an SPPS configurator that can be run as standalone programs on devices such as smartphones and tablets. This strategy is justified by the fact that the sales force would require little training and make use of devices that are becoming very popular as tools to configure the SPPS. Moreover, these tools can be used in places where there is no Internet connection (typical of rural areas in developing countries), which renders a centralized configuration server useless. However, it is well known that product configuration tends to be a time-consuming process that requires high-performance computers. Hence, one might wonder, how is that a handheld device can be used for this role?

In this paper, we apply a new approach to model the configuration of product families to the SPPS problem. This approach is based on a knowledge framework which combines a generic product structure (GPS) and a constraint network extended with design function (CN-F) model to represent product families. One important outcome of this approach is that, if a few modelling conditions are satisfied by the CN-F model, deriving product family members becomes a direct (backtrack-free) process. To show the applicability of our approach to the SPPS problem, we implemented a configurator which is capable of quite challenging solutions. The configurator was also tested on an emulator for mobile devices to demonstrate the viability of running it as a standalone program on a handheld device.

In what follows we will first make a brief review of related works. Then, we will apply our modelling approach to represent an SPPS product family and define the method for deriving its members. After that, we will present our implementation of the SPPS configurator. Finally, we make some concluding remarks.

## 2 Related Works

Constraint network (CN) has been a successful approach to deal with configuration problems [2]. However, since its proposal [3], various extensions to the classical CN model have been introduced to cope with the specificities of product configuration problems [4]. Typically, backtrack algorithms are used to find solutions for configuration problems, which tends to be a time-consuming process. To overcome the inefficiency of the search method, knowledge from the configuration domain has to be used to guide the search process [5]. On the other hand, it has been shown that under some conditions, depending on the topology of the CN model and its degree of consistency, finding solutions can be a backtrack-free process [6]. Other backtrack-free approaches rely in a pre-processing stage to map all the solutions in the design space prior to the configuration stage, when the actual problem-solving occurs [7]. Although the search for solutions is faster, these approaches cannot avoid the exponential grows of memory utilization in the pre-processing stage.

From the engineering perspective, graph grammar has been used as an approach for deriving members of product families [8]. A review on product families and methods for their design can be found in [9, 10], respectively.

### 3 The Solar-Powered Pumping Product Family

A typical SPPS is composed of a few components. At the core of the system, there is a photovoltaic (PV) array that provides power to a water pump. To improve the pump performance, a pump controller is used to condition the power and to control the pump. A float switch ( $S_T$ ) is used to turn the pump off when the water tank is full, and another switch ( $S_W$ ) is used to turn the pump off when the water level at the well is low, thus avoiding that it runs dry. The components of an SPPS are connected by wires to transmit power and control signals. The water is carried from the well to the tank through a piping system. A battery bank may be added to the system in case it is necessary to pump water at night or during heavily clouded days. The charging of the battery bank is carried out by a charge controller. However, selecting and configuring these components to meet the application requirements and to optimize the system performance is far from trivial and may involve a large solution space.

Our approach assumes that the product family has already been developed. However, the framework with the GPS and CN-F models will capture all the relevant knowledge about the product family to define precisely its solution space and to set up the process to derive its members.

#### 3.1 *The Generic Product Structure*

The GPS is a modular architecture composed of component types, which stands for classes of components. In our approach, component types belong to four possible categories: common/generic, optional/generic, common/specific and optional/specific. We say that a GPS represents the architecture of a given product family if and only if the architecture of each member of that family is isomorphic to a substructure of the GPS.

Figure 1 shows the GPS for our SPPS product family. The PV array, pump, controller, sensors, wiring and piping are considered to be common component types, i.e. they are present in every member of the SPPS product family. Otherwise, the battery bank and charge controller are optional component types. The well and tank sensors will be taken as specific component types, i.e. they do not vary among applications. Otherwise, all the other components are of the generic type, i.e. they will vary among applications and have two or more variants. It should be noted that, according to our classification, to be a common component type in the product family architecture does not imply that it is fixed. Actually, in

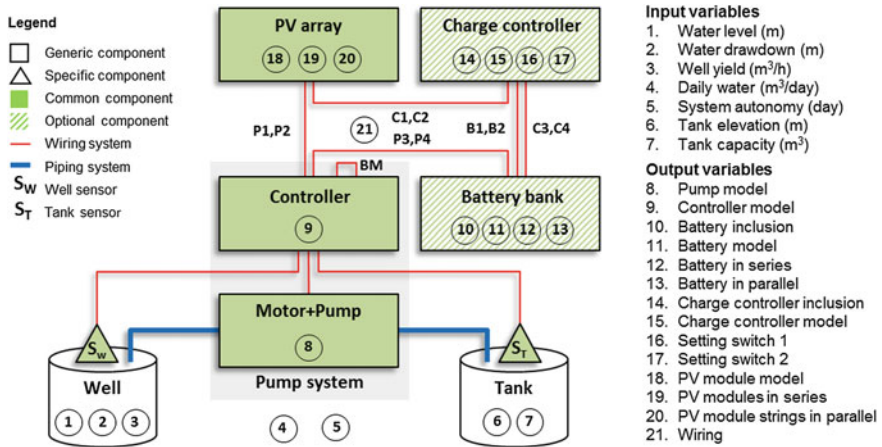


Fig. 1 GPS of the SPPS product family and the associated variable points

our example, most of the product family variability happens on the common part of the GPS.

The GPS is a useful means to delimit the design space of a product family because it constrains the possible arrangement of the components. However, it is not enough to determine completely what the valid configurations are and, in particular, what specific configuration will meet the requirements of a given application. Therefore, in our approach, the GPS is combined with a constraint network model to restrict the design space further.

### 3.2 The Constraint Network Extend with Design Functions

The CN-F model used in our approach can be regarded as an extension of the traditional CN model. It is defined by the tuple  $\langle V, C, F \rangle$ , where  $V$  is a set of variables,  $C$  is a set of constraints, and  $F$  is a set of design functions (abbreviated as  $d$ -function), such that every variable in  $V$  has at least one  $d$ -function attached to it that generates its values. In this section, we will see how these elements apply to the SPPS product family and how they complement the GPS.

**Variables.** Optional components in a product family are one main source of variability. However, most of the diversity in a product family may be due to the variability of the generic components. For example, a pump manufacture may provide a large number of pumps options, each one designed to operate optimally within a narrow window of total dynamic head and flow rate. Pumps may also be based on different pumping principles to maximize either head or flow rate. Variations among the product family members can be identified by variables and localized on the components of the GPS, but the scope of a variable can vary

widely. It may be related to one specific feature or to the component as a whole. The set of all the values that can be assigned to a variable is its domain of variation. For instance, the domain of the variable *Pump model* is defined by the set {HR-03, HR-03H, HR-04, HR-04H, HR-07, HR-14, HR-20, C-SJ5-8, C-SJ8-7}.

Since the variability of the product family is related to the generic or optional component types, only such components are associated with variables, and because these are the variables that specify the product family members, they are referred to as output variables. A special type of output variable is the inclusion variable associated with optional component types (e.g., *Battery inclusion*). These are binary variables that defines if the component is included or not in the derived product. However, variations will also be related to the application environment. For example, applications will vary in the amount of daily water needed, the water yield of the well, the capacity of the tank, if pumping is necessary when the sun is not shining, etc. These variables express the customer requirements and are referred to as input variables. Figure 1 shows the list of all the input and output variables for the SPPS example. However, input and output variables are not necessarily disjoint subsets of  $V$ , and besides these two classes, the set  $V$  may contain a subset of auxiliary variables (shown in Fig. 6), which are neither input nor output variables. For example, the variable *Total dynamic head* is defined in terms of input variables, and although it is an essential variable for the choice of the pump system, it is not used to specify directly any of the components in the GPS. Therefore, it is classified as an auxiliary variable.

**Constraints.** Constraints define how subsets of variables in  $V$  are related to each other, thus restricting the possible combinations of values that can be assigned to them simultaneously. For example, the following constraints describe how the auxiliary variable *Total dynamic head* is related to some other variables in  $V$ :

- C7 *Total dynamic head* is equal to the sum of the *Water level*, *Water drawdown*, *Tank elevation* and the friction loss of the piping system
- C8 *Total dynamic head* must be less or equal than the head of the pump system, defined by the combination of the *Pump model* and *Controller model*

To satisfy a constraint, the values assigned to the variables in the expression defining it must render the expression true. If an optional component will not be included, all constraints involving non-inclusion variables on that component can be disregarded. An assignment of values to all the variables in  $V$  such that no constraint in  $C$  is violated is said to be a solution to the CN-F model. The set of all solutions will be denoted by  $S$ . As we will argue below, solutions in  $S$  correspond to members of the product family.

**Design Functions.**  $D$ -functions have been introduced as an extension to the CN model to capture the necessary knowledge to generate the values for the variables in  $V$ . For example, Fig. 2 shows the specification of  $d$ -function F4, which generates the values for *Total dynamic head* (the dependent variable) as a function of *Water drawdown*, *Water level* and *Tank elevation* (the independent variables).

---

**F<sub>4</sub>(Water drawdown, Water level, Tank elevation)**  
**Begin**  
 1. *Geometrical head* = *Water drawdown* + *Water level* + *Tank elevation*;  
 2. *Friction Loss* = 0.05 × *Geometrical head*;  
 3. *Total dynamic head* = *Geometrical head* + *Friction Loss*;  
 4. If *Total dynamic head* ≤ the head of the available pump systems  
 5.     Then I/O (“The configuration process will be aborted because there is no available pump  
 6.         system that can overcome the total dynamic head.”);  
 7.         Abort configuration;  
**End**

---

**Fig. 2** Specification of the *d*-function F4 attached to the variable *Total dynamic head*

Actually, an important consequence of *d*-functions is the dependency relation that they establish between variables. Now, if the value generated by a *d*-function is to be consistent with the values of the variables it depends on, it must incorporate all the constraints involving these variables. We say that a *d*-function incorporates a constraint if and only if every combination of the values of the independent variables (for which the *d*-function is defined) and the value generated from them, satisfy the constraint. For example, from lines 1, 2 and 3, it can be verified that constraint C7 is incorporated by F4. However, in general, a *d*-function is not dependent on all the variables related to the variable it is attached. For example, although the variables defining the pump system are related to *Total dynamic head* by C8, F4 is not dependent on them. Note that, the condition introduced in line 4 is only used as part of the configuration control mechanism.

Input variables are attached with special *d*-functions that request the user to assign a value chosen from a delimited range of values, which may be generated dynamically. Hence, except possibly for the input variables, all variables in *V* will necessarily depend on some other variable due to the *d*-function attached to them, forming a network of dependencies on *V*.

If a set of variables is strongly coupled, they can be grouped together to form compound variable and the same *d*-function will generate the values for all them. This is the case for F12, which selects the pump system from a performance table which, for a given total dynamic head, correlates the output flux and the input power for optimal performance of the pump systems. Inclosing strongly coupled variables into the same *d*-function also prevents dependency loops between variables.

It is this capability of *d*-functions to establish acyclic network of dependencies over *V*, together with their capability to generate values that are locally consistent, that forms the basis for the backtrack-free configuration process in our approach.

Moreover, since only values generated by the *d*-functions are taken into account in the configuration process, the domain of a variable in the CN-F model is defined as the set of all values that can be generated by the *d*-functions attached to it. An important consequence of this definition is that the domains need not to be defined

**Instantiation algorithm:****Begin**

1. Following the order in  $F$ , remove the first  $f_x$  which is enabled
2. Generate a value to  $x$  using  $f_x$  and remove from  $F$  all design functions attached to  $x$
3. If  $F \neq \emptyset$
4.   Then go to 1
5.   Else go to End and return "Success"

**End**

**Fig. 3** Control algorithm for finding solutions to the CN-F model

explicitly and can be either discrete or continuous. All these properties have been used in our SPPS example.

## 4 Deriving Product Family Members

Based on our knowledge framework for representing product families, the configuration process will be divided into two stages. First, a solution to the CN-F model is found from the values of the input variables. Second, this solution is used to transform the GPS into a specific model representing the desired product family member.

As mentioned in Sect. 2, approaches base on the CN model typically use a backtracking method to find solutions. Moreover, in general, those approaches will not guarantee that a solution can be found or if this will happen in a reasonable time. However, if the CN-F model satisfies the following modelling conditions: (1) There are no dependency loops between the variables, (2) each variable has only one  $d$ -function attached to it, (3) the  $d$ -functions are defined for every combination of values of variables they depend on, and (4) every constraint is incorporated by some  $d$ -function; then, it can be demonstrated that the control algorithm shown in Fig. 3 is enough to find a solution for every consistent input to the CN-F model. This result is still valid if we make condition 2 less restrictive by allowing more than one  $d$ -function to be attached to the same variable. However, to handle this general condition, it would be necessary to introduce the concept of instantiation patterns (something we will not do in this paper). Anyway, for the SPPS example, the more restrictive condition is sufficient.

As it can be verified, the control algorithm is a straightforward one and does not admit backtracking.  $D$ -functions in  $F$  may be organized in any order. A  $d$ -function is enabled when all the variables it depends on are assigned values. The algorithm will iterate at most as many times as the number of elements in  $F$ . The demonstration that the CN-F model of the SPPS product family satisfies each of the above modelling conditions and that the control algorithm always leads to a solution will

not be given here for space reasons. However, some evidence is given in the next section.

The second stage of the configuration process consists in transforming the GPS into a specific product model based on the solution found in the first stage. To achieve this, initially we remove from the GPS the optional components that are not needed. This happens whenever the value of the inclusion variable associated with the optional component has a specific value, for instance, zero. Otherwise, the component is kept on the structure. After this cleaning process, the remaining generic components are substituted by specific ones, as defined by the values of the output variables on them assigned by the solution. The resulting model can be used for the assembly of the correspondent product family member. Moreover, this product will meet the application requirements that started the configuration process. Since both stages use direct methods, we can conclude that deriving members of the product family is a direct (backtrack-free) process.

If the product family has been properly modelled, then every solution in  $S$  corresponds to a product family member and, on the other hand, every member of the product family must correspond to a solution to the CN-F model. In this case, we can conclude that the set  $S$  of solutions to the CN-F model precisely defined the solution space of the product family, and every consistent set of requirements expressed by the input variables leads to a product family member.

## 5 Implementation of the SPPS Configurator

The SPPS configurator was conceived as a tool to support the sales force, requiring them to have only enough technical knowledge about SPPS to make some assessments at the customer site. Customer requirements may be entered all at once or requested interactively by the configurator. To avoid inconsistent inputs, the configurator performs some checks on the requirements and suggests appropriate actions to remove them. The output of the configurator is the instantiation of the product family GPS by specifying the components, their settings and arrangement.

The method for the modelling of the configuration process can be generalized into the following steps: (1) definition of the product family GPS; (2) identification of the variables; (3) definition of their constraints; and (4) specification of the  $d$ -functions. These steps have been followed in Sect. 3 for the development of the SPPS product family; thus, we will not discuss them further. However, it should be stressed that adjustments to the CN-F model may be necessary in view of the four conditions stated in Sect. 4. Notably, the options at the input variables must be restricted appropriately to prevent downstream functions to become undefined. The abort condition introduced in F4 is an extreme action that prevents the customer requirements to be outside the domain of definition of the  $d$ -function that selects the pump system. Actually, if that happens, it does not make sense to continue with the configuration process.



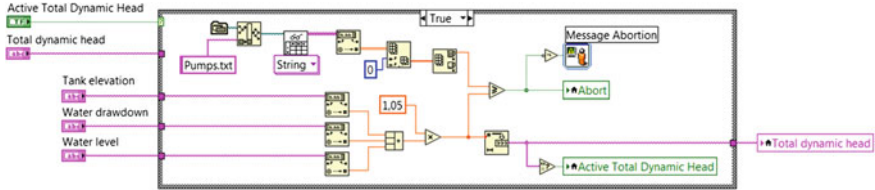


Fig. 4 Implementation of the *d*-function F4 in LabVIEW

Figure 4 shows the implementation of the *d*-function F4 (specified in Fig. 2) in the programming language LabVIEW. This is a relatively simple *d*-function. The most complex are the ones that define the arrangement of the battery bank and the PV array. The latter one, for example, calculates all possible arrangements for each PV module model available within certain limits of voltage and current and chooses the one that optimizes the power in reference to a target power. As a secondary optimization criterion, this *d*-function also maximizes the open-circuit voltage of the array.

Technical information about the available components is arranged as tables and implemented as .txt files. They are directly used by the *d*-functions. For instance, in F4 the file “Pumps.txt” is used to check for the availability of a pump system that can overcome the total dynamic head of the application. Note that, by the addition of new pumps to this table, the total dynamic head limit of the configurator can be increased.

Figure 5 shows the general algorithm for the SPPS configurator. At the centre, it can be seen the *d*-functions (numbered F1–F17), each one representing a subVI (a kind of routine in LabVIEW), with the variables to which they are attached at the right of the diagram. The variables to which the *d*-functions depend on are indicated by the connection lines from below. The arrangement of the *d*-functions in the diagram clearly reveals the dependency between variables. As it can be verified, there is no dependency loop between the variables. At the left of the diagram, it can be seen the control structure which operates in conjunction with the loop structure (the outer structure encompassing the whole program). Initially, only the first four *d*-functions will be executed. If the abort condition in the *d*-function F4 is true, there is no solution for the configuration problem and the program ends. Otherwise, the abort variable is set to false and the other *d*-functions are executed. As the *d*-functions are executed, they generate the values for the correspondent variables and are set to inactive. *D*-functions attached to variables (other than the inclusion variables) on components that will not be included are set to inactive without generating values. When all the functions are inactive (which is equivalent to  $F = \emptyset$  in the control algorithm in Fig. 3), a solution has been found and the program ends. This happens in exactly three iterations of the configurator program.

Figure 6 shows the solution of the CN-F model found by the configurator after a test case for the values of the input variables. The values of the output variables

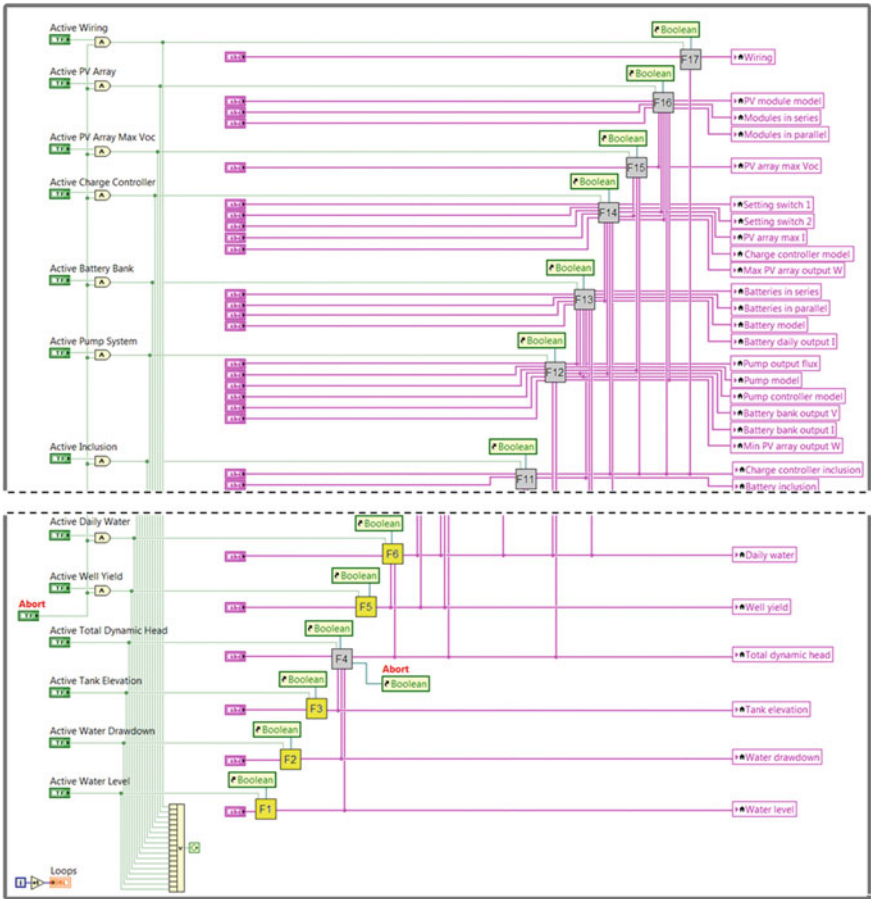


Fig. 5 Excerpt of the general SPSS configurator program implemented in LabVIEW

Input Variables													
Water level	Water drawdown	Well yield	Daily water	Tank elevation	Tank capacity	System autonomy	ASSUMPTION Insolation = 6kWh/m2d						
10 (m)	12 (m)	5 (m3/h)	1800 (m3/d)	3 (m)	45 (m3)	3 (d)							
Auxiliary Variables													
Min PV array output W	Max PV array output W	PV array max Voc	PV array max I	Battery bank output I	Battery bank output V	Battery daily output I	Pump output flux	Total dynamic head	Tank deficit	Sun deficit			
NA	1600	150	45	5	48	111	0.81 (m3/h)	58 (m)	-9.00 (m3)	>5.04 (h)			
Output Variables													
Pump controller model	Pump model	Battery inclusion	Battery model	Batteries in series	Batteries in parallel	Charge controller inclusion	Charge controller model	Setting switch 1	Setting switch 2	PV module model	Modules in series	Modules in parallel	Wiring
PS600	HR-07	1	ACM1	8	1	1	Tristar 45	ON	ON	LC120-12P	6	2	P3 P4 BM C1 C2 C3 C4 B1 B2

Fig. 6 Simplified interface of the SPSS configurator showing the solution for a test case

specify the member of the SPPS product family that meets those requirements. For an illustrative purpose, the auxiliary variables and their values are also displayed.

The program described above was ported to Windows Mobile 6 professional, ARM920T Device Emulator. Tests with the emulator showed that after all the customer requirements have been input, the time to obtain solutions was in the order of milliseconds.

## 6 Conclusions

In this paper, we have shown that the configuration process of the SPPS product family can be arranged into a direct method by the concatenation of the *d*-functions into a dataflow process. Because LabVIEW is a programming language based on the principle of dataflow, it proved to be an interesting choice to put that property in evidence.

The short response times of the SPPS configurator running as standalone program on an emulator for mobile devices demonstrate its viability as a tool for popular devices such as smartphones and tablets, without Internet connection. This also implies that there is room for the design of a much more sophisticated configurator without compromising its responsiveness on the mobile device.

Besides the performance, another conspicuous advantage of our approach is the level of modularity that can be obtained. The resulting software program is such that any *d*-function can be improved without affecting the other functions or the control structure. The same is true for the introduction of new *d*-functions. For example, we could have included a *d*-function that automatically defines the insulation level from the location of the well. Moreover, because the available components are given as .txt files, new components can be included independently from the *d*-functions. All these features will facilitate greatly the maintainability of our configurator.

Finally, it is important to note that, besides the correct configuration of the SPPS, configurators may have a major role in the automation of the total sales cycle, thus giving a further competitive advantage to the company [11]. This was outside the scope of this work. However, tasks such as the preparation of the quotations and bill of materials could have been included as additional *d*-functions.

**Acknowledgments** The authors wish to gratefully acknowledge the financial support of FINEP for the realization of this work.

## References

1. Meier, T.: Policy recommendations to improve the sustainability of rural water supply systems. IEA PVPS Task 9, Subtask 1 Report IEA-PVPS T9-11: 2012 (2012)
2. Sabin, D., Weigel, R.: Product configuration frameworks—a survey. *IEEE Intell. Syst. Appl.* **13**(4), 42–49 (1998)

3. Mittal, S., Frayman, F.: Towards a generic model of configuration tasks. In: Proceeding 11th International Joint Conference Artificial Intelligence, pp1395–1401. Morgan Kaufman, San Francisco (1989)
4. Veron, M., Fargier, H., Aldanondo, M.: From CSP to configuration problems. In: AAAI-99 Workshop on Configuration, Orlando, Florida (1999)
5. Wielinga, B., Schreiber, G.: Configuration design problem solving. *IEEE Expert* **12**(2), 49–56 (1997)
6. Freuder, E.C.: A sufficient condition of backtrack-free search. *J. ACM* **29**(1), 24–32 (1982)
7. Hadzic, T., Subbarayan, S., Jensen, R. M., Andersen, H. R., Hulgaard, H., Moller, J.: Fast backtrack-free product configuration using a precompiled solution space representation. In: International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems. Technical University of Denmark, Lyngby, Denmark (2004)
8. Du, X., Tseng, M.M., Jiao, J.: Graph Grammar Based Product Family Modeling. *Concurr. Eng. Res. Appl.* **10**(2), 113–128 (2002)
9. Jiao, J., Simpson, T.W., Siddique, Z.: Product family design and platform-based product development: a state-of-the-art review. *J. Intell. Manuf.* **18**(1), 5–29 (2007)
10. Simpson, T.W., Aaron, B., Slingerland, L.A., Brennan, S., Logan, D., Reichard, K.: From user requirements to commonality specifications: an integrated approach to product family design. *Res. Eng. Des.* **23**(2), 141–153 (2012)
11. Forza, C., Salvador, F.: Managing for variety in the order acquisition and fulfillment process: the contribution of product configuration systems. *Int. J. Prod. Econ.* **76**, 87–98 (2002)