Product Configuration over the Internet*

Jesper Møller Henrik Reif Andersen Henrik Hulgaard

The IT University of Copenhagen and *ConfigIt Software*
Email: {jm,hra,henrik}@configit-software.com

Abstract

This paper describes a product configuration tool called *ConfigIt Developer* for rapid development of websites where users can tailor products such as goods and services to specific needs. The tool supports all phases of the development of a configurator-backed website, from the modeling of the product to the deployment of the configurator on the Internet.

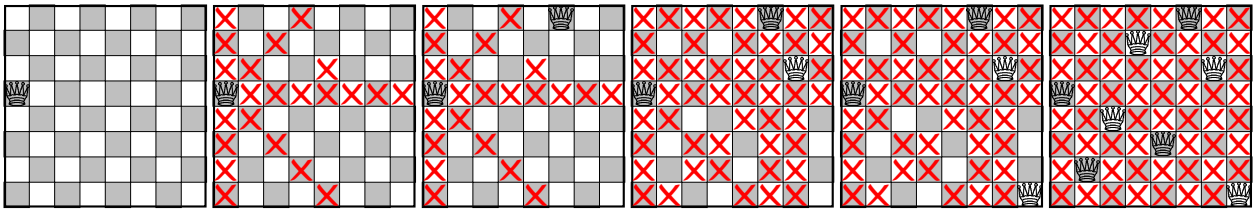
1 Introduction

Configurable products—such as computer systems, customized bikes or cars, furniture, and made-to-order factory parts—are characterized by having a set of inter-dependent parameters that the user must choose among to obtain a properly functioning product. The parameters can take on sets of values such as enumerations of values (e.g., "red", "green", "blue") or numerical values. Parameters can be real physical parts that are to be components of a customized product or attributes of a component. Parameters may also capture needs and have no direct physical counterpart but still be related to what choices are appropriate. Inter-dependencies among values are captured by a collection of rules. The rules express, for instance, constraints on which parts fit together and on the possible attributes of any given part. They can also express relationships between a parameter capturing a need and the physical components fulfilling that need.

A *product model* is a precise description of the parameters and their dependencies. The *parameters* are represented by variables $x_1, \ldots, x_n$, and the set of the possible *values* for the parameters are represented by the sets $D_1, \ldots, D_n$. The dependencies over the parameters are described by a list of *rules* $\phi_1, \ldots, \phi_m$ over the variables.

Product configuration is the task of finding a valid configuration of a product model by selecting values for the parameters $x_1, \ldots, x_n$ such that the rules $\phi_1, \ldots, \phi_m$ are fulfilled, and the user ends up with a satisfactory solution. Technically, this takes the form of a *guided search* in which the user repeatedly either inquires *a configurator* about what values are legal for the variables, or makes a choice by assigning a value to a variable. A user can be a customer who buys goods for personal use, a salesperson who uses a configurator for sales support, a technician who helps customers, or an engineer who searches for purpose-built parts for a machine. The key operation of the configurator is, given an assignment to a subset of the variables, return what values can be assigned to the remaining variables.

*Joint work between The IT University of Copenhagen and ConfigIt Software.



(a) (b) (c) (d) (e) (f)

Figure 1: The 8-queens problem.

The 8-queens problem is an academic example which illustrates how a configurator supports a guided search. The problem is to place eight queens on a chess board such that no queen is on the line of attack of any other queen. In ConfigIt Developer's product modeling language, we can describe this as follows:

```
constant
   N: 8;
variable
   col: array N of [0..N-1];
rule
  forall i in [0..N-2]:
     (forall j in [i+1..N-1] :
        (col[i] <> col[j] and
         i+col[i] <> j+col[j] and
         i+col[j] <> j+col[i]);
```

The initial inquiry to the configurator reveals that we may place a queen on any square of the chess board[1]. Assume we place a queen on A5, see Fig. 1(a). The configurator then responds that the next queen cannot be placed on any of the crossed-out squares in Fig. 1(b). Most of the crossed-out squares are obvious, except for E6 which is disabled because there are no solutions with queens on both A5 and E6. Next, we place a queen on F8, see Fig. 1(c), and the configurator responds by disabling more squares. The configurator also places a queen on G6 because all other squares in the G column are disabled and thus the user is left with no choice but G6, see Fig. 1(d). Placing a queen on H1 results in the final configuration shown in Fig. 1(e).

2 Related Work and Contributions

Many commercial tools exist for performing product configuration—for example the ILOG Configurator [4], Selectica's ACE Enterprise configurator [5], the Tacton Configurator [6], and Baan's SalesPlus [1, 9]. These tools are all based on constraint propagation techniques, originally developed in the AI community. They basically work as follows: whenever a user selects a value for a parameter, the configurator derives the consequences of this choice by searching the rules specified in the product model.

Our main contribution is a new, patent-pending technique for compiling a product model into a so-called *virtual table* that represents all solutions to the configuration problem in a compact

[1]This is not obvious—one could imagine that there were no solutions with, for example, a queen in a corner.

form. The compilation of the product model has several advantages compared to the traditional constraint propagation technique:

- High performance at runtime since the configuration problem is solved at compile time.
- Predictable response times that only depend on the size of the virtual table.
- Syntactic formulation of the rules does not influence the size of the virtual table.
- Rules and product information from databases need only be available at compile time.

The size of the virtual table depends on the structure of the product model (i.e., the number of variables and rules), but not on the number of legal solutions. For product models that appear in practice, the virtual table is usually very compact, typically less that a few hundred kilobytes. However, there is no guarantee that the virtual table is always compact. On the other hand, it is possible to prove that no representation of the legal solutions exists which is compact for all product models. This is because the Boolean satisfiability problem (which is NP-complete) can be solved efficiently given a compact representation of all solutions.

It is not possible in general to tell when the size of the virtual table explodes, or guarantee that any configuration problem can be compiled efficiently. In fact, no technology can give such a guarantee due to the NP-completeness of the configuration problem. What we can guarantee is that if it is possible to compile a small virtual table then the configurator is capable of serving the requests very efficiently.

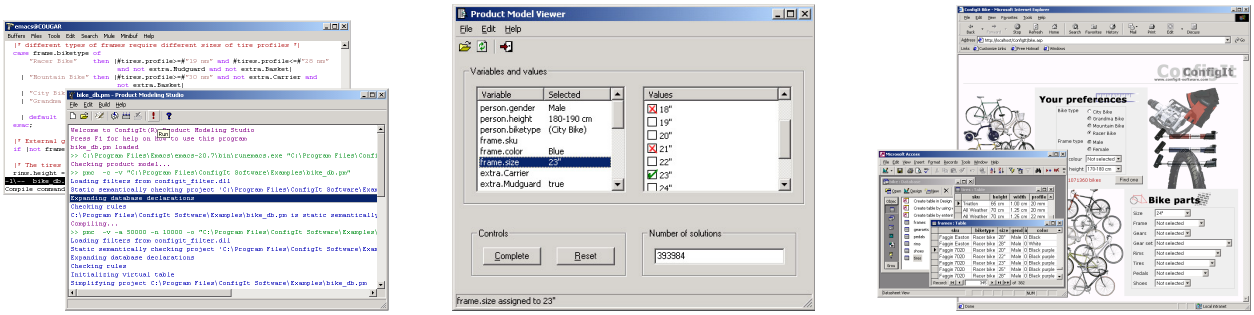
3 ConfigIt Developer

ConfigIt Developer [3] is an integrated tool environment for developing configurator-backed web-sites. ConfigIt Developer consists of four main components:

1. The *product model compiler* (pmc) for translating a product model written in PML to a *virtual table*. PML is a high-level, strongly typed product modeling language which can either be stored in a textual format or as an XML schema.
2. The *product modeling studio* (pmstudio), depicted in Fig. 2(a), provides a simple integrated development environment for editing, compiling, and viewing product models.
3. The *product model viewer* (pmview) for investigating the product model in a graphical, stand-alone configurator, see Fig. 2(b).
4. The *configuration server* which is an encapsulation of the virtual table in a COM object [7]. The configuration server makes it easy to build web pages with configurator support as for example the configurable bike depicted in Fig. 2(c).

During the compilation of the product model, the compiler finds all valid configurations and stores them in a data structure called a virtual table. Finding all valid configurations of a product model might be costly[2]. However, in practice we have found that product models can be compiled

[2]The PML language is rich enough to encode NP-complete and even harder mathematical problems. In thread with the current state of research in complexity theory, it is therefore impossible to give any formal bound on the size of the virtual table as a function of the product model. What we can say is that when the table is constructed, response times are bounded and predictable by properties of the table.



(a) Product Modeling Studio (b) Product Model Viewer (c) Configurable Bike

Figure 2: ConfigIt Developer.

in a few seconds and even very large sets of solutions end up being represented by small virtual tables. For example, we have developed a bike configurator [2] which has 127 million solutions, but requires only 29 kilobytes of virtual table.

When the product model has been compiled, the corresponding virtual table is loaded into the configuration server. The virtual table holds all the information necessary to respond to inquiries about the valid configurations. In principle, a virtual table is like a real, explicit table of all solutions: An assignment of a value to a variable $x = v$ corresponds to the removal of all records in which x does not have the value v, and obtaining the legal domains of a variable corresponds to returning all possible values occurring for a given field in some record of the table. Counting the number of valid configurations corresponds to the length of the table and so on.

A key advantage of virtual tabulation is that the resulting table is independent of the sources of the rules and the exact formulation of them. It depends only on the actual space of solutions. The runtime of the compiler might be influenced by how the rules are defined in the product model, but the size of the resulting virtual table and thus the performance of the configuration server is independent of how the model is written.

The configuration server is implemented as an NT service using COM-technology[3]. The use of the standardized COM object paradigm provides a simple interface to a configuration service, allowing a website to interact with the configurator using, for example, C++ or Visual Basic (in an ASP script). The small size of the virtual table makes it possible to download it on a client for local configuration.

4 The Product Modeling Language

Consider a bike shop that wants to sell custom-designed bikes over the Internet. The user can select among different components—frames, rims, tires, gears, and so on—but not all combinations are legal. When the user chooses a particular frame, only the rims that will fit on that frame should appear in the menu with rims and so on. A product model for the bike describes the components, their attributes and their inter-dependencies. The basic data types in PML are Booleans, integers, and enumerated types. For example, we can model the color of the bike as an enumeration:

```
type
   Color : [ Black | "Black Purple" | Red | Green | Yellow ];
```

[3]A Unix implementation is also available.

The basic types are used in record types to allow the construction of larger structured models as, for example, the frame component of the bike:

```
variable
   frame : {
      name:     Name;
      color:    Color;
      category: [ Racer | Mountain | Touring ];
      size:     [0..28];
      ext_gear: boolean;
   };
```

That is, a frame has the attributes: `name` (e.g., a stock keeping unit or an id), `color`, `category`, `size`, and `ext_gear` (whether external gear can be mounted).

Very often there will be *inter-dependencies* among the components of a product. These inter-dependencies are expressed as *rules* in PML. An inter-dependency can express *compatibility*, e.g., that the size of the rims and the tires must match:

```
rule
   rims.height = tires.height;
```

An inter-dependency can express *availability*, e.g., that a given frame is only available in red:

```
rule
   if frame.name = Faggin then frame.color = Red;
```

Finally, inter-dependencies may express *classification*, for example:

```
rule
   if frame.type = Racer then is_sportive = true;
```

Typically, rules are on one of the following forms: (1) *If-then rules* on the form `if` c `then` ϕ express that ϕ must be fulfilled whenever the condition c holds. (2) *Conflict rules* on the form `not`$(\phi_1$ `and` $\cdots$ `and` $\phi_m)$ express that the conditions $\phi_1, \ldots, \phi_m$ are conflicting (i.e., they can never be fulfilled simultaneously). (3) *Database-deduced rules* are obtained as the result of an SQL-query. If the resulting table has fields corresponding to variables $x_1, \ldots, x_k$ and the records in the table are $(r_{11}, \ldots, r_{1k}), \ldots, (r_{l1}, \ldots, r_{lk})$ then this corresponds to the formula $(x_1 = r_{11}$ `and` $\cdots$ `and` $x_k = r_{1k})$ `or` $\cdots$ `or` $(x_l = r_{l1}$ `and` $\cdots$ `and` $x_k = r_{lk})$. In other words, the table is interpreted as a formula on disjunctive normal form.

The richness of the product modeling language allows the formulation and maintenance of rules to take place in various ways depending on the particular application and skills of the persons in charge of the task. The simple if-then rules and conflict rules can easily be formulated and maintained from a simple graphical user interface resembling the configurator presented to the end user. These rules are in fact quite powerful and sufficient for many applications. Other rules, as the table-deduced rules, can be obtained from external systems—such as ERP-systems, PDM-systems, or other external data sources—and maintained from within these systems as part of maintaining other data. Writing more general rules, such as the nested `forall` rule in the 8-queens problem, might require some more knowledge of boolean logic and the product model at hand.

```
<form action="shirt.asp" method="post">
  <table>
  <tr><td>Material:</td>
      <td><% ConfigItRadioButtons "material" %>
          <% ConfigItBoolean "itches" %>Itches</td></tr>
  <tr><td>Size:</td>
      <td><% ConfigItSelect "size" %></td></tr>
  <tr><td>Color:</td>
      <td><% ConfigItSelect "color" %></td></tr>
  </table>
</form>
```

(a)

(b)

Figure 3: Web-based configurator.

5 Web Integration

To allow users to use the configurator over the Internet, ConfigIt Developer provides a library of Visual Basic functions for interfacing to the COM configuration server through the web-server. Suppose we want to develop a configurator for the following model of a shirt:

```
variable
   material : [ Cotton | Flax | Polyester ];
   size     : [ XS | S | M | L | XL ];
   itches   : boolean;
   color    : [ Blue | Green | Red | Yellow ];
rule
   if color=Red then size=L;
   if color=Blue then material=Cotton and order(size)>=order(M);
   if material=Polyester then itches else not itches;
   not (itches and color=Yellow);
```

Figure 3(a) shows how to write a configurator-backed web page in ASP for this model, and Fig. 3(b) shows the corresponding output in a browser. Each time the user performs a selection, the ASP script reloads the web page and updates the drop-down menus, radio buttons, and check boxes by making appropriate inquiries to the configuration server (not shown in the figure). The compilation of the product model into a virtual table which is loaded into the configuration server effectively separates the development of the product model and the web-integration.

6 Performance Estimation

The compilation of the product model into a virtual table, implicitly listing all legal configurations, ensures good and predictable runtime response times. By measuring the service demand D for a request to the configuration server, we can use a queue model [8] to estimate expected response time for various numbers of concurrent web-users.A good estimate for the response time r as a function of the number of concurrent users n is:

$$r \geq \max\left(nD - Z, D + \frac{(n-1)D}{1 + Z/D}\right),$$

where Z is the thinking time of the users, that is, the time between two consecutive requests from a user.

A typical service demand measured on a 450 MHz Pentium III is $D = 35\text{ms}$. With $Z = 5000\text{ms}$, the queue model gives the prediction that up to 157 concurrent users will get configuration response times within half a second and up to 171 concurrent users will get it within one second.

7 Conclusion

We have given a short presentation of the tool ConfigIt Developer. The unique feature of ConfigIt Developer is the product model compiler which precompiutes all legal product combinations and stores them in a compact data structure called a virtual table. A virtual table can be accessed through a COM configuration server, and efficiently inquired with short and predictable response times from a web page, making it easy to develop configurator-backed websites for complex products.

References

[1] *Baan SalesPlus*. See www.baan.com.

[2] *ConfigIt Bike Configurator*. See www.configit-software.com/configit/bike.asp.

[3] *ConfigIt Developer*. See www.configit-software.com.

[4] *ILOG Configurator*. See www.ilog.com.

[5] *Selectica ACE Enterprise*. See www.selectica.com.

[6] *Tacton Configurator*. See www.tacton.com.

[7] Don Box. *Essential COM*. Addison-Wesley Publishing Company, 1998. ISBN 0-201-63446-5.

[8] E.D. Lazowska, J. Zahorjan, G.S.Graham, and K. C. Sevcik. *Quantitive System Performance*. Prentice Hall, 1984.

[9] Bei Yu and Hans Jørgen Skovgaard. A configuration tool to increase product competitiveness. *IEEE Intelligent Systems*, 13(4):34–41, July 1998.