A Tentative Approach for Integrating Sales with Technical Configuration

Guy A. Narboni

Cabinet Implexe
France
email: implexe@gulliver.fr

Abstract

Sales and technical configuration occur at two different steps in make-to-order business processes for manufacturing. They follow different objectives and, above all, involve different skills. When it comes to computerizing both sales and engineering forces, what are the best options for application development? Considering the costs of updating rule-based systems in practice, we suggest a methodology to alleviate the overall maintenance problem of corporate configuration knowledge bases.

Introduction

Sales and technical configuration occur at two different steps in make-to-order business processes for manufacturing. The former when tendering, the latter when contracting. Although sales configuration comes first, it substantially depends on technical configuration. The reason behind this apparent paradox is simple: properly configuring a product generally requires a good knowledge of it.

The key importance of knowledge in configuration tasks justifies the development of Expert systems to assist users in configuring products. Software configurators typically couple a knowledge base with an inference engine that runs in the background. The sales engineer (or the site engineer) selects product parts or features that will meet the customer's requirements. Immediate consequences are drawn and incompatible choices are ruled out automatically, leading to greater work efficiency.

However, practice has revealed that knowledge base maintenance can become a critical issue, in conjunction with the high rate of change in product specifications. Since the first generation of rule-based systems used for configuration in industry [McDermott 1982], the situation, hopefully, has eased up a bit. Today's model-based systems [Junker 2006] use constraints in configuration rules as a replacement for production rules (which suffered from a lack of separation between algorithmic control and domain knowledge). Constraints state declaratively that a relation is to be maintained, without requiring specific code to do the maintenance.

Still, the cost of maintaining one knowledge base must not be underestimated. And the question becomes all the more acute when having to deal with, potentially, two.

Product versus pricing configuration

Sales configuration aims at providing a quotation for a custom product offer, i.e., basically a price with a synthetic description. Technical configuration, on the other hand, has to output a detailed view of the product specification purchased, in the form of a bill of materials (BoM) to be attached to the production order sent to the shop floor. Given that the two configuration tasks follow different objectives, are carried out at different times, in different places, by different teams (in the front and the back office), ensuring a seamless workflow from inquiries to orders can become challenging. For complex products, enterprises must be watchful not to have hiccups in the processing of the sales transaction.

When it comes to computerizing both sales and engineering forces, what are the options actually?

- using two configurators (i.e., two independent knowledge bases), with twice the development, maintenance and presumably synchronization costs?
- referring to a unique knowledge base (necessarily, the most detailed), with the risk of flooding the commercial user with technical questions that are not pertinent to pricing?

Clearly, the alternative is not satisfactory. Trying a midway between these extreme cases could yield a better solution.

An approach based on knowledge structuring

We here investigate the possibility of organizing configuration knowledge around a common core model, so as to maximize knowledge sharing (with respect to the product) and minimize semantic differences (with respect to the users). More precisely, following a methodological view much in the vein of Aristotelian classification, we take advantage of the possibilities offered by inheritance in hierarchies to extend a preliminary product description, suitable for commercial configuration, into a more detailed one, suitable for technical configuration.

For exposition, we use the small car configuration example of [Mittal and Falkerhainer 1990] which involves reasoning on conditional parts. After slightly restructuring it and introducing activity variables to explicitly indicate the participation of a component in the solution, we can freely distribute configuration rules among the various chunks of knowledge composing the new base. We then show, using two constraint models related by a subsumption ordering (one for sales, one for engineering), how to significantly reduce the number of solutions when considering only abstract (commercial) configurations, as opposed to concrete (technical) ones.

Finally, we recapitulate the assumptions made, list foreseeable difficulties in the design and management of a two-level knowledge base, and conclude on the necessary requirements on tools and technologies for generalizing this approach to problems of industrial size.

Revisiting the car model

The generic product model is described component-wise (the car, its power sub-system, the air conditioning and sunroof options). We use a feature matrix to depict it.

$$\begin{bmatrix} \text{Frame} & \ldots \\ \text{Package} & \ldots \\ \text{Engine} & \begin{bmatrix} \text{Size} & \ldots \\ \text{Battery} & \ldots \end{bmatrix} \\ \textit{Aircond} & \begin{bmatrix} \text{Type} & \ldots \end{bmatrix} \\ \textit{Sunroof} & \begin{bmatrix} \text{Type} & \ldots \\ \text{Glass} & \ldots \\ \textit{Opener} & \begin{bmatrix} \text{Mode} & \ldots \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Every component definition includes a series of functional features (parameters, with slots for atomic values) and of structural features (nested components, within brackets). The total number of parts is not fixed. Italicized names denote optional components.

The table below lists the 8 problem variables with their discrete domains, using dot notation for access paths. Implicitly, we need 3 additional 'activity' variables for controlling the selection of optional parts (booleans).

Variable	Status	Domain
Frame	required	{convertible, sedan, hatchback}
Package	required	{luxury, deluxe, standard}
Engine.Size	required	{small, med, large}
Engine.Battery	required	{small, med, large}
Aircond.Type	optional	{ac1, ac2}
Sunroof.Type	optional	{sr1, sr2}
Sunroof.Glass	optional	{tinted, not_tinted}
Sunroof.Opener.Mode	optional	{auto, manual}

Variables nested in components which are not selected are not part of the problem and therefore do not have to be assigned a value.

Note we have slightly changed the formulation by adding some structure to the original (flat) model, so that an instance looks closer to a hierarchical bill of materials. We thus define a car configuration as an edge-labeled tree of feature-value pairs, like:

$$\begin{bmatrix} \text{Frame} & \textbf{sedan} \\ \text{Package} & \textbf{standard} \\ \text{Engine} & \begin{bmatrix} \text{Size} & \textbf{small} \\ \text{Battery} & \textbf{small} \end{bmatrix} \end{bmatrix}$$

From the point of view of satisfiability, the two knowledge representations are equivalent [Geller and Veksler 2005]. Structuring, however, is key to compositionality and reusability.

There are 14 configuration rules, expressed at the car level. 8 'activity constraints[1]:

- Sunroof option included if Package = luxury
- Sunroof option included if Package = deluxe
- Sunroof option excluded if Frame = convertible
- Aircond option included if Package = luxury
- Aircond option included if Sunroof.Type = sr1
- Aircond option excluded if Engine.Battery = small and Engine.Size = small
- Sunroof.Opener option included if Sunroof.Type = sr2
- Sunroof.Opener option excluded if Sunroof.Type = sr1

6 'compatibility' constraints:

- Package = standard excludes Frame = convertible
- Package = standard excludes Aircond.Type = ac2
- Package = luxury excludes Aircond.Type = ac1
- Sunroof.Type = sr1 and Aircond.Type = ac2 excludes Sunroof.Glass = tinted
- Sunroof.Opener.Mode = auto and Aircond.Type = ac1 forces Engine.Battery = med
- Sunroof.Opener.Mode = auto and Aircond.Type = ac2 forces Engine.Battery = large

The configuration problem boils down to finding a satisfying assignment. It is easy to see that Package = luxury and Frame = convertible are incompatible user choices, whereas the request Package = luxury and Engine.Size = small has several solutions.

[1] Due to variable clustering, there is no need for the original 'Always Require Variable' activity constraints.

Adding gradations to knowledge

A general requirement for sales and technical configuration integration is the following [SAP 2006]:

> If you produce complex products, such as industrial machinery, you particularly need a multilevel product configuration tool that provides sales with an easy-to-use tool to generate preliminary bills of materials based on customer specifications and provides engineering with a power-user version to create a detailed configuration for the final quotation.

Acknowledging the differences in expectations concerning interactive tools —as well as the differences in product expertise, we argue that we can make provisions for two levels of use (for sales representatives and site engineers), while keeping a unique knowledge base for reference.

Depending on the context, we are used to reason with different levels of details. Even during technical configuration, the BoM tree is only developed to a 'macro' level: the lowest configurable one. Later on, when collecting detailed information for requirements planning, BoM items corresponding to composite parts are deterministically expanded into a 'micro' list of parts for creating a production order.

This strict separation of concerns cannot apply when attempting to draw a boundary between sales and technical information, for a simple reason: configuration decisions remain interrelated. No choice can be made in total isolation. So, the remaining options for setting the right priorities and lowering the amount of technical details are twofold:

- information hiding (declare some intricate features 'private'): this idea is easy to implement and preserves completeness of solving. However, it does not reduce the model's complexity (it may indeed obscure its behaviour).
- information dropping (deactivate part of the conjunctive constraint net): this is a more radical move which de facto commands model simplification, at the expense of completeness. Interestingly though, this approximation is sound in the sense that the relaxed model will never reject a technically satisfiable configuration request.

In the sequel, we mainly emphasize this second approach (a rather trivial 'abstract interpretation' of the problem's constraints), showing how it can be applied in a conservative way, i.e., without throwing configuration knowledge away.

Methodology

The formalism of order-sorted feature trees [Aït-Kaci and Podelski 1991] allows us to equip our models with the declarative inheritance mechanism popularized by object-oriented programming.

Considering a concrete class of component, we can always split its definition of into two:

- a first half is inherited from a super-class (the genus), to which the informations of major importance are moved
- the second half contributes to the definition of a sub-class (the species), which restores the details of minor importance (the differentiae).

For the concrete product model, nothing is lost in the process (all structural and behavioural properties, inherited or supplemented, are enforced). The conjunction of the definitions of the super-classes now makes up an abstract model.

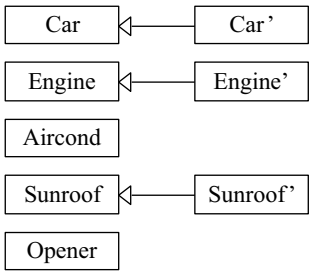
Furthermore, based on the class inheritance ordering we can partially order instances. This corresponds to subsumption ordering (considering how much information a solution provides). Intuitively, a technical configuration should enrich a sales configuration. Indeed, a sales configuration should specificy the information that must be preserved in any technical configuration extension.

Let us summarize our working hypotheses at this stage:

1. The sales and engineering views can agree on a common product model.
2. This pivotal model is the technical configuration model (which intentionally defines a valid BoM tree, with constrains in conjunctive form).
3. The technical configuration model is a refinement of the sales configuration model (which abstracts away configuration details).
4. The sales configuration model is a relaxation of the technical configuration model (which implements necessary but not sufficient conditions)
5. Every technical configuration instance is subsumed by a sales configuration instance.
6. The atom-valued features of the sales (resp. technical) configuration model that are made public encompass all the basic criteria used for expressing functional (resp. high-level and low-level) requirements.
7. The product knowledge base is organized into a two-level hierarchy which relates and integrates the two configuration models.

Results

To illustrate our point, we consider that informations of lower relevance are the car's battery and sunroof opener. Since we should not focus on them in the first place, we provide abstract and detailed descriptions for the classes Engine and Sunroof. The resulting hierarchy is:



The scaled-down abstract model[2] (Car, Engine, Aircond, Sunroof) allows us to reduce by three fourth the complexity of the configuration problem.

Model	detailed	relaxed
Variables	8+3	6+2
Constraints	14	9
Solutions	450	120

Incidently in this case, there is always at least one technical combination of battery and opener items suitable for extending a solution to the relaxed model into a solution to the original problem. Every partial configuration synthesized is therefore a valid sales configuration that can be used for an initial quotation.

Perspectives

The car example shows that sales configuration (without involving price calculations) can be viewed as a relaxation of technical configuration. However, the fact that the stand-alone sales model maintains configuration consistency is due to chance. In the general case, a global validation check would involve running the fully featured model once, as an oracle, in autonomous mode. Note this requirement calls for automatic completion capabilities.

Yet, the methodology sketched in this paper does not require 'on-the-fly' inheritance mechanisms [Mailharro 1998]. One can always flatten the hierarchy in a pre-processing step. Since each abstracted component has a unique hier, there is no ambiguity about the concrete type of parts.

We just claim that a hierarchical decomposition of the product model should be maintained in the master version of the configuration knowledge base. This unique source could be used for generating two variant configurators, for sales and engineering purposes. Using a single knowledge base is of vital importance for avoiding the well-known maintenance bottleneck.

In the real world, decisions for seperating high-level and low-level knowledge could be more awkward and might involve a mix of feature hiding and feature relegation. An upfront investment in knowledge (re-)engineering is obviously mandatory for properly designing a two-level configuration model.

For the developer, an immediate positive effect could be to clarify matters concerning the user-interface definition. For the end-users of the sales configurator, the expected benefits should be better acceptance, better understanding and better performance. For the site engineers finally, it could avoid misapprehensions in the data handed over, and consequently facilitate the tracking of customer order changes.

As noted above for implementing this approach, knowledge editing facilities are more needed than new runtime solving capabilities. In that respect, a lack of 'knowledge independence' in a configuration tool can definitely be spotted as an impediment to large scale experiments.

References

Aït-Kaci, H., and Podelski, A. 1991. Towards a meaning of LIFE. In Proceedings of the Third International Symposium on Programming Language Implementation and Logic Programming, 255-274.

Junker, U. 2006. Configuration. In *Hanbook of Constraint Programming* (Rossi, van Beek and Walsh eds.). Elsevier B.V.

Mailharro, D. 1998. A classification and constraint based framework for configuration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 12(4): 383-397.

McDermott, J. 1982. R1: A rule-based configurer of computer systems. *Artificial Intelligence* 19: 39-88.

Mittal, S. and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In Proceedings of the Eigth National Conference on Artificial Intelligence (AAAI), 25-32.

Geller, F. and Veksler M. 1995. Assumption-based pruning in conditional CSP. In Proceedings of the Eleventh International Conference on Principles and Pratice of Constraint Programming (CP), 241-255.

SAP 2006. The challenge to achieve perfect order management. SAP A.G. white paper. www.sap.com

[2] The delta is: minus 1 ternary plus 2 binary variables; minus 3 activity plus 2 compatibility constraints (marked -).