Rule-Based Configuration
ABSTRACT

Organizations that sell build-to-order and engineer-to-order products often require a product configuration application to handle the complexities of ensuring proposed products are buildable and accurately priced. Buildable means that all of the selected options will fit together properly and be compatible with each other when the product is built or the service is provided. Organizations, such as those in the computer hardware/network, heavy trucks, aerospace and insurance industries, that do not have a reliable means of submitting buildable orders to their manufacturing or fulfillment groups incur substantial costs in terms of rework and additional, unpleasant communication with prospects. There are dozens of well-known product configuration application vendors that facilitate accurate orders. However, only the elite vendors offer applications that are capable of handling the complex configuration requirements of organizations that propose products and services with thousands of options and totaling into the millions of dollars. This document describes often-overlooked complex configuration concepts that any organization seeking a product configuration application should consider.
CONTENTS

Introduction ..................................................03
   The Product Configuration / CPQ Application
   The Configuration Engine
   The Data Management Application
   Data Updates
Advanced Concepts ...........................................XX
   Guided Selling
   Analytics
   CRM Agnostic
   Renewals / Automated Quoting
   Engineered Options
   Automated Decision Making
   Flexible Data Model
   Lookup Tables
   Rule Dependency
      Simple Boolean Rules
      Shared Expression
      Generated Rules
      Determining Dependencies
   Caching Initial Values
   Application Extensions

Conclusion ..................................................XX

We are FPX ..................................................XX
INTRODUCTION

Product configuration is the process of specifying a product or service, adding options, and determining if the finished product is valid. Product configuration also includes calculating a price based on the selected options, pricing factors such as region, and customer-specific discounts and pricing agreements. The values and calculations that determine the offer price can be rule-driven in the same way product-option compatibility and availability are rule-driven.

The Product Configuration / CPQ Application

The image below shows a common approach to product configuration where option categories are presented as expandable folders, and options for the various groups are listed in a table with a selection control or quantity input box. A composite illustration of the product is updated each time the user makes an option change.

A product configuration application is most often used by a salesperson and others involved in the sales process. The salesperson will select a base product and then add options to meet the requirements of the customer. Later, during the approval process, engineers may access the configured product to approve a customized option and management may need to look at special pricing. If the application also has the ability to generate a quote/proposal, then it is often called a “Configure Price Quote” (CPQ) application.
The Configuration Engine

The configuration engine validates each product and option selection made by the end user of the CPQ application. When the user selects a product or option, the engine may:

- Evaluate the relevant rules to determine if other options need to be selected or deselected.
- Apply appropriate pricing based upon rules, calculations, customers, tiers, or attributes.
- Evaluate rules that may cause messages to be displayed. For example, when the user selects an option, there may be a message that alerts the user of an available package of options at a discounted price.
- Recalculate values such as weights, prices and costs.
- Update a product illustration.

The engine ensures that a configured product is always valid. In other words, the product does not contain incompatible options, the product is not missing required options, and the pricing is correct.
The configuration engine consumes data sets produced by a data management application. The data management application is where product managers define the rules for option compatibility, define when the selection of a product or option should cause other options to be included automatically, create rule-driven messages for the CPQ user, and manage pricing.

Large organizations typically have master data sourced in other applications. Pricing, and pricing agreements for specific customers, as well as product and option order codes, descriptions and specifications, likely exist across multiple sources. The data management application has built-in and/or accompanying data conversion and import capabilities to acquire data from those sources. Data entry personnel can then augment the imported data with business rules, messages and images to prepare it for deployment to the CPQ application. In some cases, after the initial data load, the data management application is considered the master rather than relying on data imports.
Data Updates

The product managers and pricing administrators publish data sets as often as needed. Some organizations provide updates every week due to price changes or specification changes. The organization needs to determine how to handle in-progress quotes that contain configured products created from a previous data set. The CPQ application should be able to handle mass-updating all quotes or simply allow the CPQ users to apply the updates if desired. For some organizations, applying a price increase to a quote in the final stages of a sales cycle would be detrimental. Applying a critical engineering change, however, could save time later when the quote becomes an order.

The configuration engine is invoked to apply a new data set to an existing configured product. During this process, the engine can handle instructions for replacing options that are no longer available, revalidate the configuration, recalculate the pricing, and report the changes made.
ADVANCED CONCEPTS

So far, this document has described capabilities found in applications offered by many CPQ application vendors that facilitate accurate orders. The rest of this document describes complex configuration concepts offered only by a small subset of vendors. These concepts, that are part of what makes complex configuration possible, are often overlooked during a vendor evaluation process. Further, because of the complexity involved, there is risk in using applications provided by vendors that do not have mature, well-tested, code. For example, as a CPQ user configures a product with millions of possible option combinations, and complicated pricing rules, the configuration engine potentially needs to re-evaluate hundreds of rules and calculations each time the user selects an option. The configuration engine needs to determine whether cached values should be used (to ensure good application performance) or whether new values need to be produced. The number of evaluations required due to one option change can be staggering. In advanced configuration, the application isn’t simply configuring a computer, it is configuring an entire network or data center. It isn’t just configuring a ventilation system, it is configuring the environmental control system for an entire football stadium. Only a robust, well designed application can handle these requirements. Poor code results in each transaction taking longer with the result being unacceptable performance, or worse, errors in the configuration and/or pricing.

The less mature the code, the greater the risk for:

- Poor application performance due to inefficient evaluation of rules and calculations.
- Errors in costs and prices due to a failure to execute the appropriate calculations.
- A failure of application extensions to work properly, especially after application upgrades. An application extension is a customization that adds functionality.
- Application crashes resulting in downtime.
- A need to abandon the application completely.
Guided Selling

Guided selling is an interaction that educates prospects about a set of products or services and captures input to deliver the optimum selection. Typically, it involves eliciting the prospect’s needs and then making recommendations that are specific to the prospect’s situation. The online interaction is analogous to an in-person experience with a salesperson who knows how to ask the right questions to guide buyers toward the right product and make them feel confident about their purchasing decisions.

The CPQ application must have the ability to present questions to the user, and then use the answers to determine the next set of questions to display and narrow down the list of recommended best-fit products and services. The product experts should be able to author, in the data management application, the questions and the resulting behaviors.

Product experts should also be able to author messages that are displayed to the CPQ user at appropriate times during the product configuration process. For example, if any component of a package is selected, but the package is not selected, a message can be displayed alerting the CPQ user that a package is available that contains the option just selected plus additional options at a special price.
Analytics

As your organization’s salespeople configure products and generate quotes, the related transaction data is saved to a database. Over time, this becomes a valuable pool of information that includes product configurations that succeeded in past deals, where the products were sold, types of customers, profit margins, and engineered options. The CPQ application must allow users to access and use this “big data” for predictive analytics.

For example, one way to use predictive analytics is in the area of advanced product search. The CPQ user can type a search phrase such as a product name or desired option features and retrieve a list of ready-to-go pre-configured products. Behind the scenes though, the engine assembled that list based on attributes important to the user’s organization. For example, there might be a preference to rank products highly that were quoted in successful, profitable deals, have a high customer satisfaction rating, and are likely to result in add-on business. CPQ users like this because it reduces or eliminates the need to configure a product (that is, to add options) and organizations like this because the proposed products have a proven history of meeting customer needs. Your organization must be able to define settings and apply weights to tune the search results.
CRM Agnostic

Every successful organization grows and changes and thus some of the systems used by those organizations will also change. The ERP and CRM applications that might have met the needs of an organization with 500 employees will be inadequate when it reaches 5,000 employees. Unfortunately, too many CPQ application vendors have tied their configuration engine and CPQ application to a specific CRM application or platform. When the organization outgrows a CRM application, it often means it has outgrown its CPQ application. This can present a huge loss of investment. The premier CPQ application vendors know that their application (CPQ, configuration engine, and data management) must work with any CRM application. The key here is having an architecture that facilitates integration, and a platform that runs independently, rather than being tied to a single CRM vendor’s platform. While some vendors may brag about being 100% native on a certain platform, ask them what it will cost to transfer the CPQ application if your organization decides to port to a more robust CRM system.
Renewals / Automated Quoting

Many organizations, especially those in the insurance and software industries, have a renewal process for an existing plan, subscription or license. The process typically involves generating a new quote from a list of customer assets (existing products and services) and making replacements and upgrades as needed. Without a CPQ application, the renewal process is time consuming and resource heavy, requiring days of administrative work across multiple departments for each renewal.

Products and services change over time, especially rates, so as a new quote is developed, multiple departments need to provide input. In insurance, the actuarial, underwriting, and sales processes can take days to complete. The CPQ application automates the renewal process by examining the assets and generating a quote that contains the new products, services and pricing.

When a renewal process begins (for example, it might be triggered on a specific date) it acquires assets and uses them to generate a new quote. In the illustration above, the current asset, XYZ100, is acquired and the configuration engine then generates a configured product that matches the existing asset. However, new pricing is applied and, if there are any option changes, the configuration engine makes those changes as well. The CPQ application must have data mapping capability to enable this to work.
Engineered Options

Complex configuration often requires the CPQ user to capture information needed for a customer-specific option or service, and the user needs the ability to at least estimate a price. This could be a list of materials and hourly labor rate estimates, or the creation of an option based on an existing option in the data.

The mature CPQ solution facilitates this through template options (ad hoc parts) and the ability to alter options that already exist in the data. A template option allows CPQ users, during product configuration, to create new option instances with attributes appropriate to what the organization provides, and then include those options on a product just like any other factory option. An alterable option allows CPQ users, during product configuration, to modify an existing option in the data so that it meets the needs of the user’s customer. Template options and alter options can trigger workflow processes to ensure the appropriate engineers review the options before a proposal is sent to a customer.

Automated Decision Making

A primary task of the configuration engine is to prevent the CPQ user from creating an un-buildable product. As the user configures a product, the configuration engine may add options automatically based on the user’s selections. For example, the selection of a performance package would cause the configuration engine to select all of the options included in that package (and also calculate a special “package price” for those options). Later on, the user may select another option that renders the performance package incompatible, thus the configuration engine must fix that by deselecting all of the options included in that package. Further, the user may have selected other options that now become incompatible. Thus, there can be user-selected options and system-selected options. The configuration engine needs the ability to make the right decisions in prioritizing what gets deselected to fix incompatible options. Some organizations need to allow incompatible options to be selected, at least temporarily. Other organizations need to allow incompatible options but only if they are user-selected. The configuration engine must have the ability to handle these requirements.
Flexible Data Model

There is no one-size-fits-all approach to complex configuration. The system requirements are as varied as the products and services sold by large organizations. For example, an organization that sells aircraft considers the weight of each option to be important, while an organization that sells health insurance is more concerned about complex calculations. The mature CPQ solution facilitates these variations through a flexible data model. Each organization must have the ability to extend the data model by adding objects and custom fields without the need to modify the application code. Further, the experienced CPQ application vendor will have a library of industry-specific template data models that can be used as the starting point for a new project.

Lookup Tables

The goal of a CPQ application is to enable salespeople to be as self-sufficient as possible when preparing a quote. This requires providing access to an immense amount of data, not just the data needed to accurately configure complex products, but also to provide customer-specific pricing, inventory, or even city tax rates. Some of this data changes more frequently than the product configuration data, so it must exist in lookup tables that the configuration engine can access when needed. By having this data separated from the product configuration data set, it can be updated more often and even by a different business group.
Rule Dependency

In the illustration below, the tire (Tire_10AA) has a rule that is true when the 16-inch rim has been selected. Thus the tire is available for selection only when that rim has already been selected. The rim is available only when the Car_150 model has been selected. Thus the availability of the tire depends on the rim which in turn depends on the car model selected.

Now consider a product with hundreds of selected options, thousands of selectable options, and millions of possible combinations. One change to the product configuration can cascade into many rules and calculations being re-evaluated. The configuration engine must be able to handle these evaluations in the proper sequence and efficiently (that is, only evaluate the rules and calculations that require evaluation). If it doesn’t, some changes can require several seconds to complete.
For each subsequent option selection, rules are checked to determine which remaining options can be selected (compatible) and are visible (available).

The most common configuration relationship is compatibility, which involves a combination of factors that determine whether or not an option is available for a particular configuration. If an option is currently incompatible, an Incompatible Option icon appears next to the option. If an option is unavailable, the option is not shown in the user interface.

A rule can determine when a message or an image is displayed. A rule can determine what price to apply to an option. A rule can also determine what warranty text blocks are to be used later in the generated proposal.

Experienced CPQ application vendors understand that the personnel responsible for getting product and service data out to the sales channels don’t want to incur a development effort every time there is a need to change a business rule or price calculation. Intelligent data management systems use simple logic syntax rather than complex programming languages. This allows product experts to quickly make the changes they need without first needing to get a degree in programming.
Simple Boolean Rules

Rules such as \(((\text{OptA or OptB}) \text{ and } (\text{OptX or OptY or OptZ}))\) only depend on the specific instances that they reference. In this case, the rule has dependencies of OptA, OptB, OptX, OptY, and OptZ. When the selection status of one of those instances changes, the configuration engine reevaluates the rule.

The result of the rule (true/false) is stored in a cache. When the result of the rule is later needed, the configuration engine pulls the result from the cache rather than evaluating the rule. By caching results, the configuration engine doesn’t need to reevaluate every rule and calculation each time the CPQ user makes a change. This results in substantial performance gains.
Shared Expression

A shared expression is an instance that represents a rule or rule segment. The data management application user creates a shared expression when a rule segment is used repeatedly in other rules. Suppose, in our rule above, the rule segment (OptA or OptB) was used in hundreds of rules. We could create a shared expression named *AorB and put (OptA or OptB) in its rule attribute (also called a field). Then we could use *AorB in any other rule. Our original rule above would become (*AorB and (OptX or OptY or OptZ)).

By using a shared expression in the rule (*AorB and (OptX or OptY or OptZ)), we have partitioned the dependencies into two different rules. When one of OptX, OptY, or OptZ changes selection state, the shared expression *AorB does not need to be reevaluated. In this small example, this wouldn’t represent a noticeable performance gain. However, imagine if the shared expression was lengthy and used within many rules. When the CPQ user makes a change to OptA or OptB, *AorB is reevaluated. Rather than evaluating the rule segment hundreds of times (once for each rule it is contained in), it is only reevaluated once. The result of *AorB has its own spot in the results cache. After reevaluating *AorB once, the new result is compared to the previous result. If it has changed, then our larger rule, and any other rules containing *AorB, are reevaluated.

Another benefit of using a shared expression is the ability to change the rule in one location and have that change reflected in all the rules that reference the shared expression.
Generated Rules

A generated rule is similar to a shared expression, but the generated rule is created by the data processor, not the data management application user. The processor, which reduces the product configuration data into a compact dataset, may recognize that the rule segment (OptX or OptY or OptZ) is repeated in enough rules that it makes sense to pull it out into a shared expression. It will do that and, at the end of processing, our large rule will actually be evaluated as (*AorB and **XorYorZ). Note that this is invisible to the user and a completely automated process.

Determining Dependencies

Each rule, calculation, and string expression is assigned a list of dependencies by the data processor. Dependencies define when a rule or a cached expression is to be reevaluated by the engine.

An option instance contains attributes such as ID, description, and price. Some of those attributes might point to a list of other instances and a rule on those other instances determines which instance to use. For example, OptA’s price attribute (PriceList) might point to several other instances that contain price data. Those price instances all have a rule attribute and the first instance with a true rule, would be the price used. The price might exist in an attribute named CalculatedPrice.
A rule in another instance can acquire the value of the current price for OptA by chaining to the value \(\text{(OptA.PriceList.CalculatedPrice)}\). The dependencies are generated according to the content of the expression. For the most part, it is from the attribute value chains. During data processing, the chain is parsed and dependencies are added as the chain is traversed.

Tracking dependencies and ensuring they are handled in the proper sequence requires an advanced configuration engine. Without it, real-time evaluation of configured products with hundreds of options and calculations would not be possible or, at the very least, lack acceptable response times.

**Caching Initial Values**

When a CPQ user starts to configure a product with a just-activated data set, the configuration engine loads that data set and immediately evaluates all of the rules to determine the initial state of the configuration. The creation of an initial state is a one-time event, provided the data set stays loaded. The configuration engine must have the ability to cache the initial configuration state and retain it for as long as the data set remains in memory. Since all new connections to the data set can use the saved initial state, system performance is improved, especially when dealing with data sets that have a large number of rules. The current date will always be set into the configuration so that the date-dependent rules provide a correct result.

**Application Extensions**

Every large organization has a unique set of system requirements to support its unique processes, integrated systems, and way of selling. The CPQ application vendor must have the ability to customize its application to accommodate these requirements. One indicator of this capability is an API to the configuration engine. While this type of customization requires a developer, and is typically a services engagement, it is often a necessary part of the implementation of a configuration solution for large organizations, despite what some vendors might claim. However, the presence of an API, and years of experience, means these customizations often can be created and tested in weeks.
**CONCLUSION**

This document describes a sample of the complex configuration concepts offered only by a small number of CPQ application vendors. Typically, these vendors have been around for a decade or more and have the been-there-done-that experience needed to think through the most efficient ways to solve these complicated issues while still providing reliable performance.

As CPQ application vendors work with increasingly complex business rules, larger data sets, more integration, and more application extensions, only those with an appropriate application architecture, experienced developers and proven management will succeed. No CPQ application vendor is immune from having to address periodic performance issues as application capabilities expand. A successful vendor is one that has the platform, resources and know-how to quickly address those issues and then move on to the next big thing.

**WE ARE FPX**

In 1983, we pioneered sales force automation. Then, we invented Configure-Price-Quote®. Now, our Predictive Applications™ streamline sales of complex product lines, for greater accuracy and improved efficiencies, in less time.

We help companies confidently and securely manage sophisticated business processes in the Cloud. We impose structure on Big Data, harness it for insights and make it actionable at all points of the interaction process. With our applications, businesses achieve tangible success by simplifying their sales processes. When companies are easier to buy from, they sell more – and increase customer satisfaction, trust, and brand loyalty.

*Learn more at FPX.com.*