Constraint Definition Language in Oracle Configurator - A Holistic Perspective

Krishnan Sundaresan
Fujitsu Consulting, Inc.

Introduction

Constraint Definition Language (CDL), in releases 11i10 and beyond, symbolizes the constraining relations between the items in a model as statements, thereby engendering the definition of complex rules for configuration management. This paper elucidates the various facets of CDL and provides a framework for its implementation, by illustrating it with an example. Concurrently, CDL is juxtaposed with the interactive rules definition used in earlier releases, thus enumerating its benefits and limitations.

The rules that define the product structure are created using the Oracle Configurator Developer (OCD) responsibility and CDL may be construed of as the mechanism that enables the rules definition. The relationships that can be defined in CDL encompass Logical, Numeric, Comparison and Property-based compatibilities. CDL does not support the creation of Design Charts or Explicit Compatibility.

Oracle Configurator – An Overview

The Bills of Material (BOM) for Assemble to Order (ATO) and Pick to Order (PTO) Models need to be defined to represent the product structure accurately. The Model BOMs represent the collection of choices that a user can select from, at the time of configuring the product. Logical grouping of these choices are denoted by the Option Classes, which have their own BOMs. A commonly used example to illustrate this is the Personal Computer. The PC would be the model within which there are several choices to choose from viz. the 19” monitor, 60 GB memory etc. Monitor, Memory etc. would be created as Option Classes with their own BOMs (the components being 19” monitor, 60 GB and so on).

Having created the Model BOM, in order to define rules that drive or exclude certain selections, the BOM needs to be imported into the Oracle Configurator schema (CZ). Once imported, CDL enables creation of these rules that depend on the product design and engineering. A User Interface (UI) needs to be created as well, that displays the choices. Subsequently the rules and the UI need to be published to the calling application that would be accessed by the end user, at which time the rules come into play. The calling application may be Oracle Order Management, iStore, Quoting or a custom application.

Any change to the BOM requires refreshing the BOM to the CZ schema, redefining rules (if required) and republishing the model rules and UI. It is important to note that only the items that had been defined as “optional” on the BOM would be available in CZ for us to be able to define rules. The components on the BOM that are mandatory (based on the product design) would not be available to base our rules on (this would be intuitive when we consider that these mandatory “included” items on the BOM would be a part of the resulting configuration, regardless of the other user selections). Additionally, new items may be defined just in CZ that enables creation of rules. These would typically be the “logical” selections that the user makes like pressure range, color etc. The framework for Oracle Configurator is depicted in Figure 1.

Modeling a BOM for Configurator requires considerable expertise in product engineering and end customer requirements and is beyond the scope of this paper. Suffice to say that the resulting Model BOM should be simpler to maintain, scalable to meet future product enhancements, engender the best runtime performance and should be based on principles and practices that can be leveraged for new product introductions.

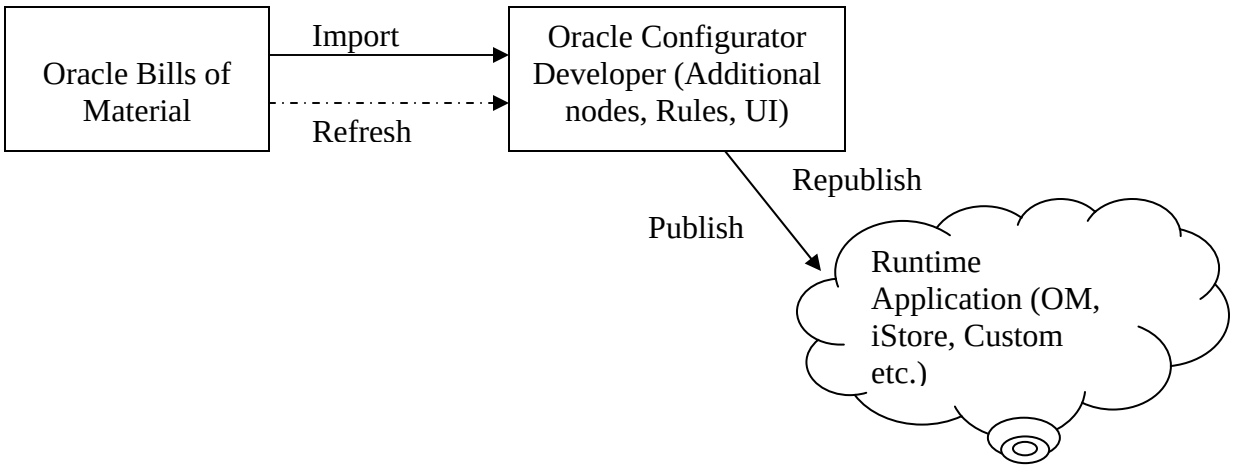


Figure 1 Framework for Oracle Configurator

Configurator Rules

Configurator rules are defined using the OCD responsibility and are based on the product structure. For instance, there may be a requirement to add certain items to a configuration when the user selects a particular option. Creating a rule to accomplish this would avert the necessity of the user having to know every item that goes with a selection. Or the requirement may be to exclude a set of items (from a different option class) when a particular option is chosen. This makes the product and the application a lot more "intelligent" while the user is not concerned with how the product is designed or manufactured.

The types of rules that may be defined in OCD include Numeric, Logic, Comparison, Compatibility and Configurator Extensions. Logic rules are defined to denote logical relationships between the various items/elements in a model. Numeric rules denote the quantitative relationships viz. selection of one item drives a specific quantity of another. Compatibility rules define what other items are allowed in a configuration when another is selected. Explicit, property-based and Design charts are types of Compatibility rules that are used in specific scenarios. Configurator Extension (functional companions until release 11i9) lets us create custom programmatic rules based on business-specific or product-specific conditions that can't be handled using the other types (engineering calculations like maximum stress allowed based on a selection, based on certain item attributes etc.).

The Building Blocks of CDL

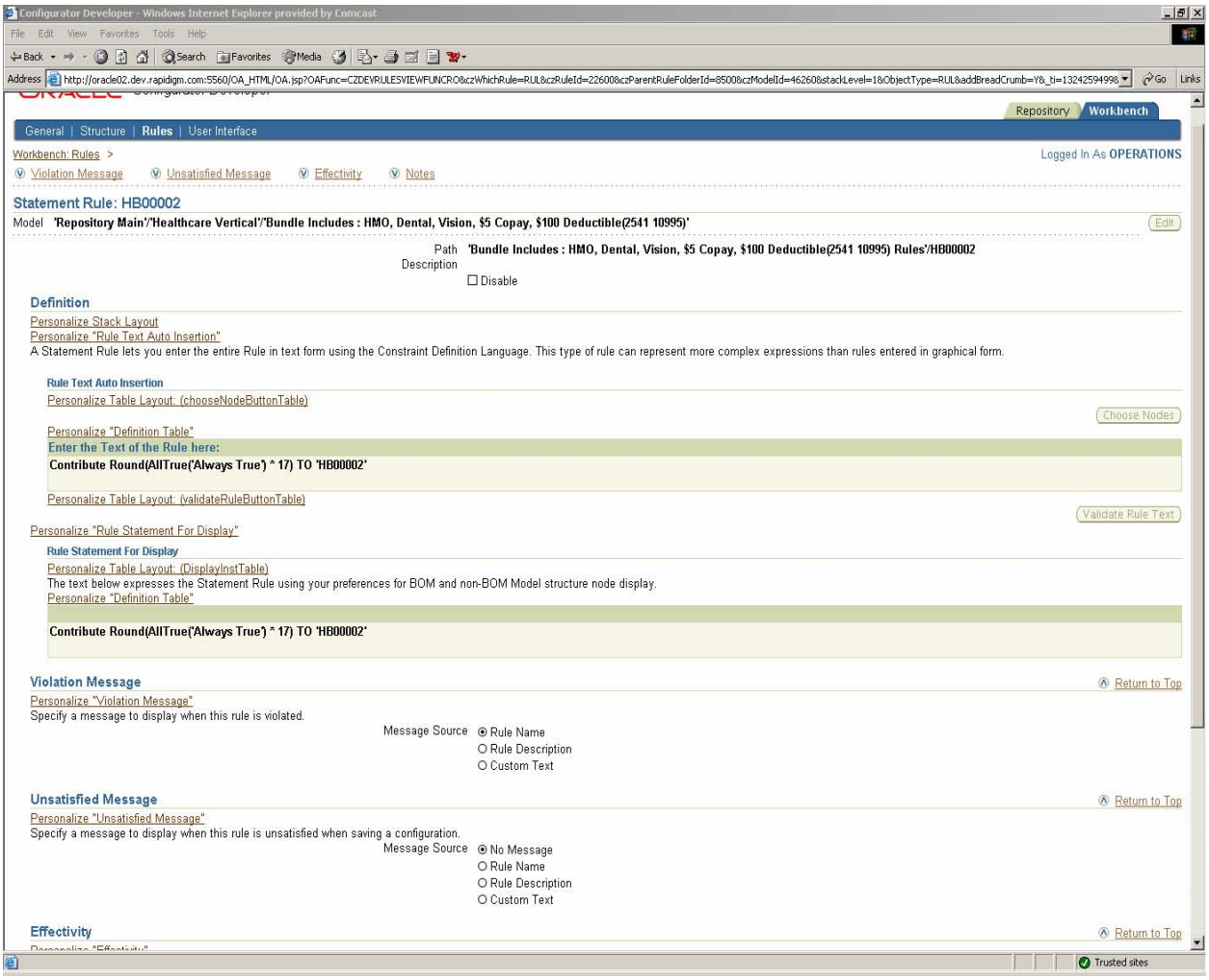
While CDL offers greater flexibility and versatility, it is imperative to have clarity on aspects like the intended purpose of the rule, the participants and expressions that would help meet the objectives. The participant nodes and properties would decide the kind of CDL statements (Explicit and Iterator, more on this later) and the stability of the Model structure (frequency of changes) would decide whether to use nodes or to use properties while coming up with the rules.

The starting point is the Model structure definition and it should be finalized based on the actual product behavior, the engineering and manufacturing considerations, usability and maintenance aspects. The objective is to keep the BOM Model structure simplified, maintainable and easy to comprehend and use.

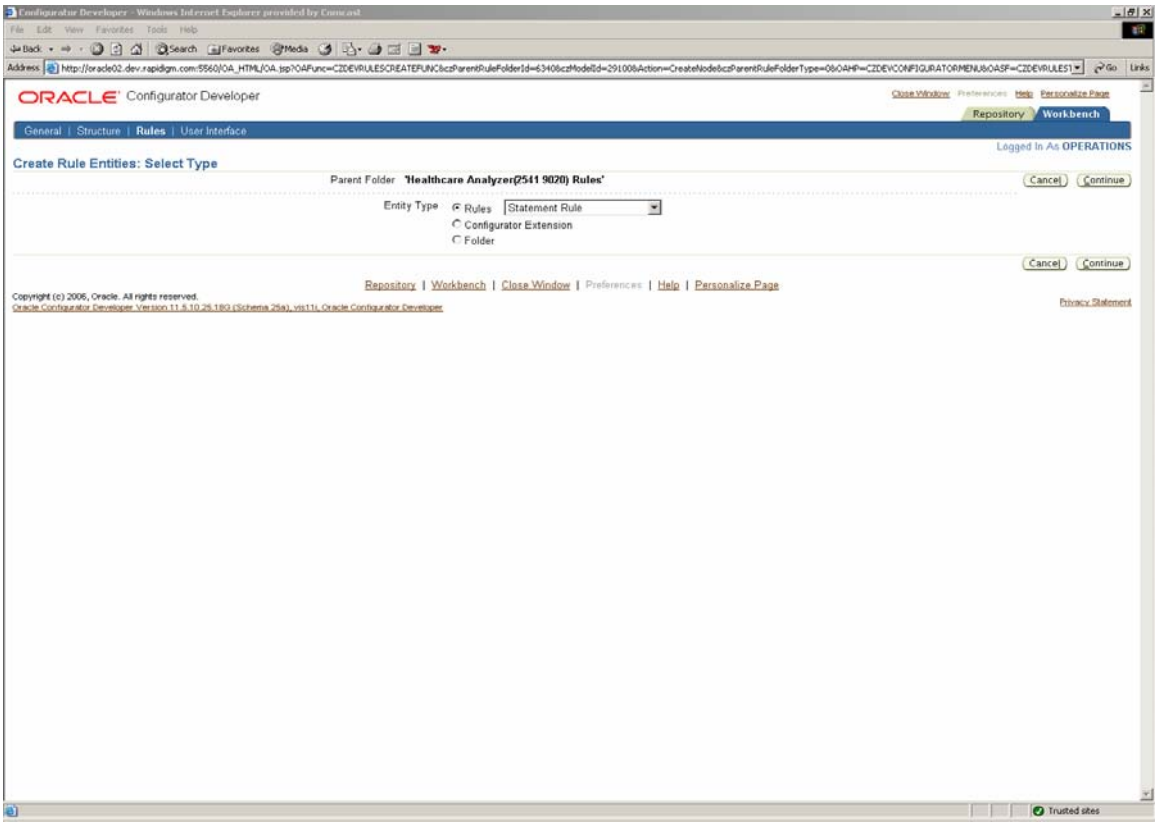
Statement rules that are written in CDL can be used to express multiple relations and operands in a single rule with a single rule being sufficient to denote both sides of the expression. There are specific keywords that need to be utilized and syntaxes followed while using CDL to define the rules. These include some of the relationship "bridges" that were being used in earlier releases (until 11i9) viz. implies, defaults, contributes to etc.

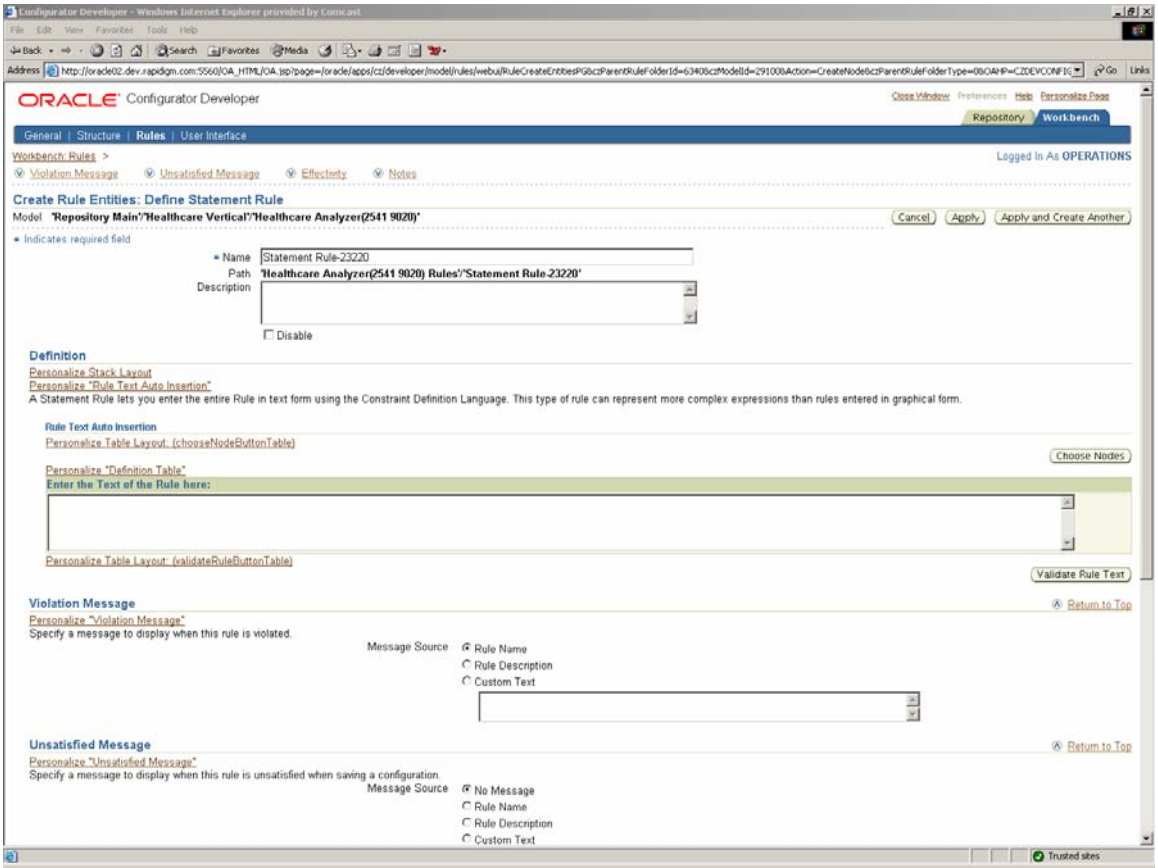
At its core, a CDL rule encompasses the basic rule definition, the corresponding statements, quotation marks, case sensitivity and comments. The rule definition includes its name, description, effectivity and usage. The statement forms the kernel and indicates what it is that the rule does. For instance, it may default a couple of items based on another item or contributes to a specified quantity of a different item etc. Effectivity and Usage are not mandatory elements of the rule and these can be defined only if the participating nodes in the rule had been created in CZ. BOM models and nodes cannot have effectivity or usage defined since they get controlled from the BOM. A violation message may be defined that displays when the rule violation occurs. It is important to note that while the participating node names are case-sensitive, the keywords and keyword operators are not. Quotation marks are used when a node name is same as the keyword. Additional notes may be added, as required.

A sample of the rule defined using CDL is given below.



Creating a new statement rule that utilizes CDL is straightforward. The Model should have been imported into CZ and the rule's intent needs to be formulated clearly. The following screenshots illustrate how the rule needs to be defined.


Configurator Developer - Windows Internet Explorer provided by Comcast
ORACLE Configurator Developer
Repository
Workbench
General | Structure | Rules | User Interface
Logged In As OPERATIONS
Create Rule Entities: Select Type
Parent Folder 'Healthcare Analyzer(2541 9020) Rules'
Cancel
Continue
Entity Type
Rules
Statement Rule
Configurator Extension
Folder
Repository | Workbench | Close Window | Preferences | Help | Personalize Page
Copyright (c) 2006, Oracle. All rights reserved.
Privacy Statement
Trusted sites


Configurator Developer - Windows Internet Explorer provided by Comcast
ORACLE Configurator Developer
Repository
Workbench
General | Structure | Rules | User Interface
Workbench: Rules >
Logged In As OPERATIONS
Violation Message
Unsatisfied Message
Effectivity
Notes
Create Rule Entities: Define Statement Rule
Model 'Repository Main'/'Healthcare Vertical'/'Healthcare Analyzer(2541 9020)'
Cancel
Apply
Apply and Create Another
* Indicates required field
* Name Statement Rule-23220
Path 'Healthcare Analyzer(2541 9020) Rules'/'Statement Rule-23220'
Description
Disable
Definition
Personalize Stack Layout
Personalize "Rule Text Auto Insertion"
A Statement Rule lets you enter the entire Rule in text form using the Constraint Definition Language. This type of rule can represent more complex expressions than rules entered in graphical form.
Rule Text Auto Insertion
Personalize Table Layout: (chooseNodeButtonTable)
Choose Nodes
Personalize "Definition Table"
Enter the Text of the Rule here:
Personalize Table Layout: (validateRuleButtonTable)
Validate Rule Text
Violation Message
Return to Top
Personalize "Violation Message"
Specify a message to display when this rule is violated.
Message Source
Rule Name
Rule Description
Custom Text
Unsatisfied Message
Return to Top
Personalize "Unsatisfied Message"
Specify a message to display when this rule is unsatisfied when saving a configuration.
Message Source
No Message
Rule Name
Rule Description
Custom Text
Trusted sites

Explicit Statement and Iterator Statement Rules

In Explicit Statements, the node and model participants in a rule need to be identified overtly and applies to a particular model. The rule execution is confined to just these participants and not to any other node in the model. Several keywords may be used in these rules such as IMPLIES, CONSTRAIN etc. A rule like X IMPLIES Y is very specific and applies to just X and Y. Other nodes in the model would not be considered by this rule.

In contrast, the Iterator statement rules utilize grouping of participants to which the rule can be applied, based on properties. So a participant with the same properties would be included in the rule even though the rule doesn't identify the participant explicitly. They come in handy when the same rules need to be maintained for several nodes or when the properties are subject to frequent changes. An example of such a rule would be FOR ALL IN. In a way, the Iterator statements operate as a query that fetches the nodes and then applying the conditions of the rule.

The Iterator statement rules complement the Compatibility based rules by including the components/nodes and then applying the rule. While the compatibility-based rules exclude certain nodes based on the conditions. They may be used in conjunction with one another.

Rule Expressions

Expressions form an inherent part of the CDL rule and consists of an operator and two operands. Until release 11i9, these were designated as Advanced Expressions that were used to define rules. For example, consider the following statement

X AND (Y OR Z)

This statement has two expressions – (Y OR Z) and the entire statement itself, with the operators being AND and OR. The same statement may be broken down into two statements as follows

X AND Y

X AND Z

Rules that indicate (X AND Y IMPLIES A) and (X AND Z IMPLIES A) may be aggregated into a single rule with the foregoing single expression.

In addition, mathematical expressions for calculation and comparison and property-based comparisons may be used in a rule.

(X AND Y) * (Y * 1.5) CONTRIBUTE TO Z

This rule indicates that if X and Y were chosen, it would result in Z for a quantity of 1.5 times the entered quantity of Y.

Defining the expressions form a key component of the Configurator Developer and involves both the product and application knowledge. One of the commonly used scenarios is to convert the metric units to the imperial units. When a customer selects an option in say, Meters, it could be internally converted to Feet using a mathematical expression in CDL rules. Structuring the expressions efficiently and accurately would enable creation of simpler rules that are easier to maintain.

CDL Keywords

As the name indicates keywords are predefined words within CDL that are used in rule statements to denote a specific relationship. A sample list of keywords would include CONSTRAIN, FOR ALL IN, WHERE and the like. Each keyword has a specific meaning and use of a keyword, on certain occasions may be optional. For example, the CONSTRAIN keyword is not mandatory when writing certain constrain statements.

The statements

CONSTRAIN X DEFAULTS Y and
X DEFAULTS Y

are equivalent and indicate the same relationship between X and Y. The CONSTRAIN keyword is used for logical relationships between rule participants and should include one of the operands – IMPLIES, NEGATES, DEFAULTS, REQUIRES or EXCLUDES.

Numeric rules utilize the keyword CONTRIBUTE...TO. If a quantitative relationship exists between two or more nodes, the rule would include the mathematical relationship.

CONTRIBUTE X TO Y

is a simple rule that indicates that if X were chosen, it would contribute to one of Y. CONTRIBUTE…TO is a powerful tool when used for conversions.

COMPATIBLE…OF is the CDL notation for property-based compatibilities and is used when a node selected by the user is deemed compatible with other nodes in the model. For example

COMPATIBLE X OF Y

indicates the compatibility of X with Y.

FOR ALL IN is an iterator keyword that would return a range of nodes satisfying the criteria.

FOR ALL X IN (1, 2, 3)

implies that all of the X nodes that are part of the set 1, 2 or 3 will be returned.

CDL and Non-CDL rules – A comparison

This section compares the rules written in CDL and those that are not. It is possible to maintain rules in non-CDL formats that accomplish the same objectives. CDL largely aids in creation and maintenance of rules that are complex and hence a little complicated to maintain in the non-CDL mode. CDL rules replace Advanced Expression rules that were used in pre-11i10 releases. A brief comparison with pre-11i10 versions is warranted due to basic architectural differences in 11i10 and beyond, for Configurator Developer as a whole.

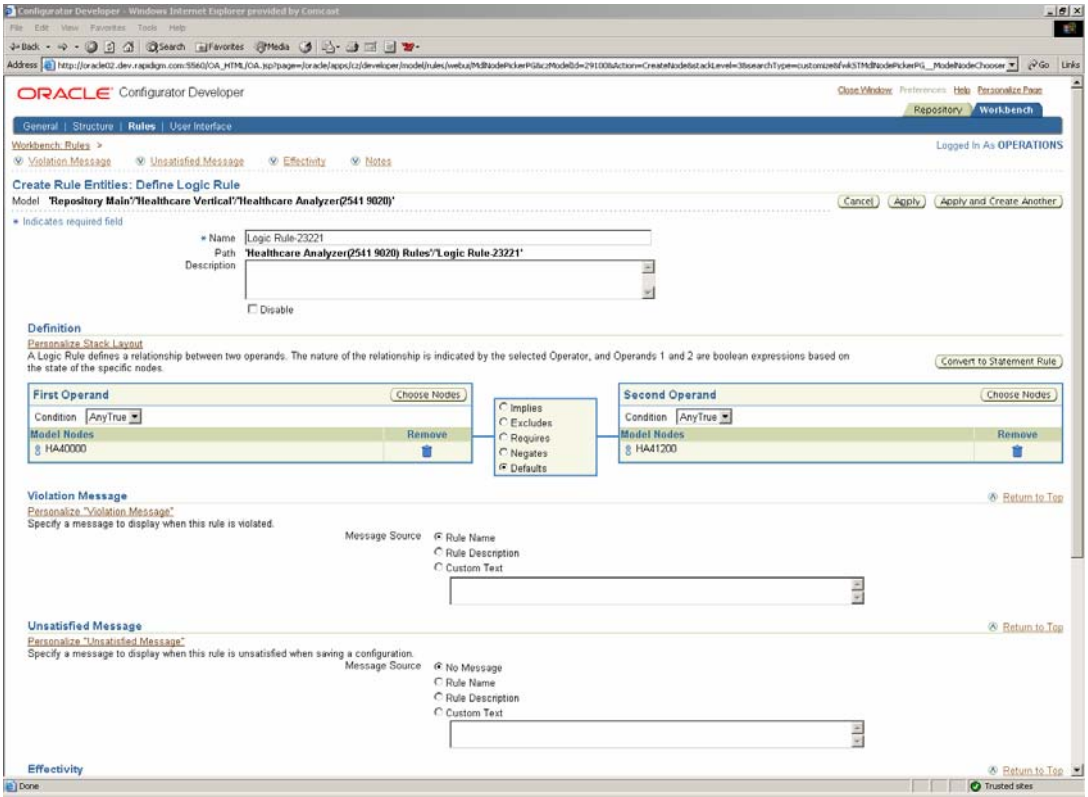
Whether to use CDL or not would depend on factors such as rule complexity, volume of non-BOM components, the nature of rule itself and the like. For e.g., CDL does not support Design Charts and so, if a rule necessitates the use of design charts, CDL is not an option. For rules that don't involve many nodes and are strictly numeric or logical (selection of X defaults Y or X contributes to 3 of Y etc.), non-CDL rules may prove simpler. However, CDL statement rules confer several advantages over non-CDL rules and should be considered in earnest.

Let us consider a simple BOM ATO model with two option classes.

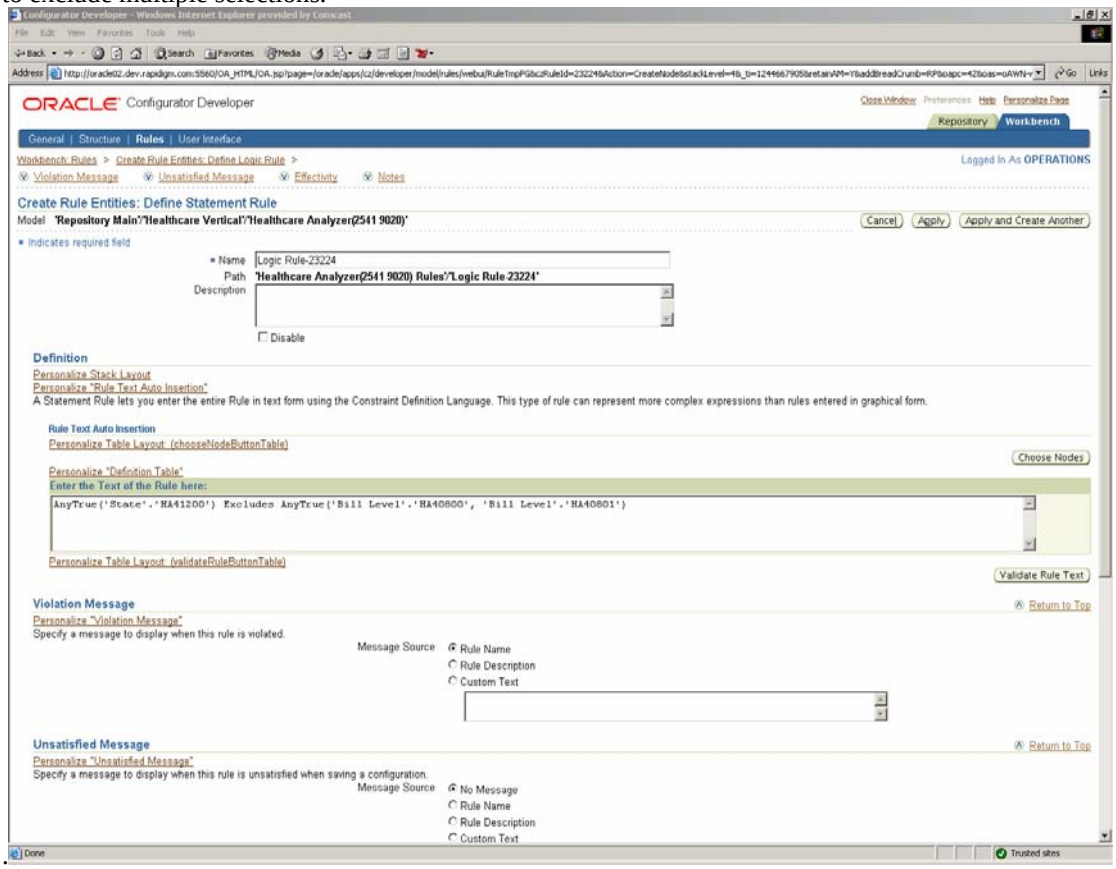
M – Model
- X – Option Class
  - X1 – Option
  - .
  - .
  - .
  - X10 – Option
- Y – Option Class
  - Y1 – Option
  - .
  - .
  - .
  - Y10 – Option

The product structure for M implies that when X1 is chosen, Y2 thru Y6 are excluded. Also X2 defaults Y7, all the time. This product structure and the rules can be designed in the CDL and non-CDL formats. Both the non-CDL formats are illustrated below.

The screenshot for the second rule X2 defaults Y7 is shown below.



This is not very different from the pre-11i10 releases. But as illustrated in the screenshot below, CDL has been used to exclude multiple selections.



Implementation Considerations

As indicated earlier, several factors need to be considered while deciding on CDL. Where the rules involve the Advanced Expressions (Boolean operators), CDL is mandatory. It would be a safe assumption to say that CDL is mandatory from 11i10 forward since almost all models and rules involve Advanced Expressions.

The ease of maintenance and efficiency in the rules definition would dictate how the rules need to be structured. For example, instead of creating two different rules like (X AND Y) defaults Z and (X and A) defaults Z, it would be prudent to create a single rule that states X AND (Y OR Z) defaults Z.

When upgrading from pre-11i10 to 11i10 and beyond, the Advanced Expression rules would be converted automatically to CDL statement rules. Also, the AnyTrue and All True operands are not available for numeric rules from 11i10 and need to be considered. Functional Companions get upgraded but the concurrent program "Migrate All Functional Companions" needs to be run for retaining them as Configurator Extensions. When the rules' logic is current and active, the logic is regenerated and the model is auto published. However it is ignored if the logic is out of date. Rules that utilized the CONSUMES FROM operator are upgraded. But the CONSUMES FROM operator is not a valid one from 11i10 and so to the CONTRIBUTE integer feature * -1 to resource statement is utilized in its place, to accomplish the same objective.

The implementation considerations would vary widely depending on the number of models, the type, number and complexity of the rules and their intended usage. Also, the runtime application would be a factor. For e.g., an ebusiness suite application like Order Management has its specific setups and requirements while a custom web application may be able to utilize custom pricing logic with its additional requirements. The foregoing discussion is indicative of the nature of issues that need to be dealt with and is not prescriptive.

Conclusion - Integration Touch points

Configurator, at its face value helps in letting the user select a valid combination of choices that results in a valid product that can be built and shipped to the customer. Once the runtime application invokes Configurator and the product is configured, there are several business decisions and software applications that it needs to flow through. For instance, the Available To Promise (ATP) and Pricing integration decide how the product is priced and promised for customer delivery. If the configured product were sourced from a vendor or a subsidiary, it would necessitate integration with the purchasing applications. How and where the inventory controls are maintained for the configured item would necessitate tighter integration with materials management. If extended warranties or service contracts are sold for the configured product, it implies that they need to be tracked in the installed base. Customer facing documents like Order Acknowledgements, Invoices etc. would have to be made compatible for the configured products. In short, Configurator touches almost every business function and the implementation needs to be viewed accordingly.

About the Author

Krishnan Sundaresan is a Principal Consultant with the Oracle Practice at Fujitsu Consulting, Inc. He has over 8 years of experience and has played a senior role in multiple Oracle ebusiness suite implementations in a broad range of industries. His involvement has included analysis, process redesign, applications setup, upgrades and user training. Krishnan holds a Bachelors Degree in Mechanical Engineering, a Masters in Industrial Management and is a Certified Supply Chain Professional from APICS. He can be reached at Krishnan.Sundaresan@us.fujitsu.com.