CPQ Sample Model

FBS\_SSC\_CA

Contents

[Keywords, Key Concepts 2](#_Toc475561194)

[Functional Description 2](#_Toc475561195)

[Description of Modeling techniques 4](#_Toc475561196)

[Guided Selling questionnaire 4](#_Toc475561197)

[Selecting Servers 4](#_Toc475561198)

[Adding Software and Services 6](#_Toc475561199)

[Similar technique is used for adding Software instances. 6](#_Toc475561200)

[Linking Software to Hardware, and Services to hardware and software 6](#_Toc475561201)

[Counting and Setting Line item quantity 7](#_Toc475561202)

# Keywords, Key Concepts

|  |  |
| --- | --- |
| Key Word | Description |
| Counting | Tallying, counting, accumulating values |
| ADT’s | Use of abstract data type pointers to represent relationships between solution components |
| NonPart Instances | Adding content without using a bom (bill of material) |
| Guided Selling | Rudimentary guided selling illustrated here. Other, better, examples may be available at the CWG website document share. |

# Functional Description

Model for a solution containing hardware, software, and services and employs a form of guided selling. Also allows for user to override the values recommended.

The model is intended to configure a data center, supply servers supporting an application with various user types. Based on the number of each user type specified, the model will generate the number of servers required – with the appropriate memory, CPU speed and quantity of CPU’s.

**User input:** Specify number of users for one or more user type.

**The model reacts by:** provides a list of potential servers to satisfy the needs of each user type specified.

**User input:** Select the server name to use;

**The model reacts by:** generates the number of servers required. These are blade systems – a Blade System contains a set of up to 14 individual servers. The appropriate number of blade systems is generated, each containing the appropriate number of servers; and each individual server will be configured with the appropriate memory and CPU’s. The users only input was the number users, and selection from a set of potential servers. Everything else is done by the model. Of course it does not NEED to be automated, users can override system default behavior if desired. See Memory Override below

**User input:** User can also select software and/or services. Either can be associated with servers by selecting servers listed in the “Runs on HW” cstic on Software instances, or in the “Serves” cstic on Services instances.

The association can also be made from the hardware side via cstics “Runs SW” and “Served by”.

**The model reacts by:** the line item quantity is updated for the Software instance and Service Instance based on the number of servers associated with them – if the software runs on 15 servers, the line item quantity for software is set to 15. That is controlled by the model, and if that is not the desired approach, it does not need to be modeled that way.

The user can **override the memory** values on any Blade System by setting the cstic SPEC\_MEM to ‘User’ (by default it is set to ‘System’). When set to ‘User’ the memory field is opened for update. Specify a new value and that amount of memory will be allocated evenly across all the servers in this specific blade system. There is no impact on other blade systems in the configuration.

Overriding other default behavior.

The model was defined to automatically link all software and all services to all server instances. That can be overridden by changing the ‘Auto Assign’ characteristic on any instance – server, software, or service. When changed, assignments are withdrawn and can be reassigned manually.

# Description of Modeling techniques

## Guided Selling questionnaire

The root instance of the configuration contains a set of prompts/questions through which the user specifies their business needs. In this case, one simple questions about volume (number of users per user type), gathers all the info needed. Based on the answer to that question, the model determines which types of servers are appropriate.

The fields accepting user input for number of users are

* STD\_SELFSERV\_USERS
* SCM\_SALES\_USERS.

The logic associated with those fields is identical, so let focus on just one of them - STD\_SELFSERV\_USERS.

Right click on the field and select “find references”, you’ll see it is used in three constraints:

* DEMAND\_STD\_SELF
* USER\_MASTER\_DATA
* VISIBLE\_INST

In constraint **VISIBLE\_INST**, you’ll see the STD\_SELFSERV\_USERS field is used to control visibility of two other fields. If no value is specified, then the list of server names is hidden, and so is the list of pointers to servers already added to the configuration

Constraint **USER\_MASTER\_DATA** is used to create an instance in the configuration to hold data specific to a user type. In this constraint, see the find\_or\_create command, and right click on the class name User\_MASTER\_DATA to find references to it. See the constraint “GET\_USER\_MASTER\_DATA”. This constraint pulls data from a variant table and stores it in the USER\_MASTER\_DATA instance. So the 2 constraints work together to get data related to any user type for which a number of users has been specified. Then in other constraints, if user specific data is needed, the User\_Master\_Data instance is referenced.

**DEMAND\_STD\_SELF** computes the ‘demand’ – required number of CPUs, CPU speed, and amount of RAM to support the type and number of users specified.

## Selecting Servers

After the user has entered the number of users in either or both ‘number of users’ fields, the field containing the list of servers is made visible. Where did that list of value come from and does it vary – is it dependent on anything else, like perhaps the user type?

The server name selection field is STD\_SELF\_SERVER\_NAME, find references to that characteristic. These are the constraints referencing that characteristic:

* VISIBLE\_INST
  + we talked about this already
* SERVER\_MASTER\_DATA
  + Same exact techniques as used for USER\_DATA
* DMN\_SERVER\_AND\_CPU\_SPD
* COMPUTE\_REQ\_SERVERS
* STD\_SELF\_INST\_1
* STD\_SELF\_INST\_N

DMN\_SERVER\_AND\_CPU\_SPD restricts the domain of both Cpu\_Speed and Server\_Name values. It does it by using a field computed in constraint DEMAND\_STD\_SELF based on User\_type.

The DMN\_SERVER\_AND\_CPU\_SPD constraint restricts CPU Speeds to only those greater or equal to the speed required for this user type:

?S.**domain** ACTL\_STD\_SELF\_CPU\_SPD >= ?S.REQ\_STD\_SELF\_CPU\_SPD

Then it also uses that restricted domain to restrict the allowed Server Name values using a variant table:

**table** T\_SERVER\_DATA

(SERVER\_NAME = ?S.**domain** STD\_SELF\_SERVER\_NAME

,PROCESSOR\_SPD = ?S.**domain** ACTL\_STD\_SELF\_CPU\_SPD

)

COMPUTE\_REQ\_SERVERS computes the number of servers required to meet the memory demand, and the number of servers needed to meet the required number of cpus. Then it compares those 2 values. The number of servers required is the greater of those 2 values.

STD\_SELF\_INST\_1 and STD\_SELF\_INST\_N work together to create the required number of servers (just calculated in COMPUTE\_REQ\_SERVERS.) The technique used here is commonly used, and is therefore very important.

STD\_SELF\_INST\_1 says “if it is true that there is at least 1 (more than 0) server required, then it is also true that there is a Server with instance number = 1.”

STD\_SELF\_INST\_N says “if it is true that a number of servers is required, and a server exists with an instance number < the numver required, then it is also true that there is a Server with instance number 1 greater that the existing one.”

To illustrate, assume 3 servers are required. STD\_SELF\_INST\_1 will create a server instance with SSC\_INSTANCE\_NUM = 1.

STD\_SELF\_INST\_N runs because its objects pattern is matched. There is a number of servers required (3) and a Server instance exists with instance\_num =1 (which is < 3). So it makes sure there is another Server instance with SSC\_INSTANCE\_NUM one greater than the one found. And that constraint runs for each new Server instance created, Except not for the one whose instance\_num = 3. (only for instances with instance\_num < 3)

## Adding Software and Services

Software is added by selecting values in field “SOFTWARE\_SELECT”, and Services are added by selecting values in fields INST\_SERVICE\_SELECT (Installation service) or MAINT\_SERVICE\_SELECT(Maintenance Services). These are multiValue fields.

Constraints CREATE\_SERVICE\_INST adds service instances. The constraint will run once for each value in the multivalue field, and using the values in the field to set a cstic value on the Service instance ensures each instance is Unique.

**constraint** CREATE\_SERVICE\_INST {

**objects**:

?S **is\_a** (300) FBS\_SSC\_CA

**condition**:

?S.SERVICE\_SELECT **specified**

**restrictions**:

**find\_or\_create**

((300) SERVICE,

**with** SERVICE\_PROFILE = ?S.SERVICE\_SELECT;

IS\_PART\_OF\_SD\_SOFT = ?S;

SERVICE\_IN\_SOL\_ADT = ?S)

**explanations**:

"CREATE service instance. multiValue will find\_create for each value."

}

Note, that we did not set SERVICE\_PROFILE on the Service instance in that find\_or\_create statement, then only one Service instance would be created, and it would be “justified” in the engines truths maintenance system once for each value in the SERVICE\_SELECT field.

## Similar technique is used for adding Software instances.

## Linking Software to Hardware, and Services to hardware and software

While each item added to the solution is a component of the solution, there are other relationships between the components. Relationships are expressed by using using ADT’s. ADT’s function as pointers to other instances.

See constraints

* LINK\_SW\_HW, LINK\_HW\_SW
* LINK\_HW\_SV, LINK\_SV\_HW
* LINK\_SV\_SW LINK\_SW\_SV,
* AUTO\_ASSIGN\_SW, and AUTO\_ASSIGN\_SV

to see how these are linked.

Each of the “LINK\_...” constraints ensures that if item A points to item B, then item B also points back to item A. For example, LINK\_SW\_HW ensures that if a Software says it runs on a Server, then that Server says it runs that Software.

The AUTO\_ASSIGN… constraints use a Switch (AUTO\_ASSIGN\_TXT) to determine whether the system should automatically establish these relationship – that is assume that all software runs on all hardware, or all software and hardware are ‘serviced by’ all services in the configuration, OR whether no such relationships are assumed. If no relationships are assumed, then the User may assign the relationships manually by making selections in the ADT characteristic fields.

## Counting and Setting Line item quantity

The model includes a “reference characteristic”. Reference cstics allows the model to reference information that exists on the sales document (such as customer numer, country, sales organization, etc) and for the model to set values of fields on the sales document, such as Line Item Quantity. The reference characteristic in this model, STOP\_MENGE, is used to set the Line Item Quantity.

**characteristic** STPO\_MENGE {

**names**

**EN** 'Component quantity',

**DE** 'Komponentenmenge'

**numericLength** 13 **decimalPlaces** 3

**negativeValues**

**reference** **table** 'STPO' **field** 'MENGE'

}

To illustrate how this could be useful, we’ve assumed that a Software item’s quantity depends on the number of things it runs on. So if 1 server runs the software, item quantity is 1; but if it runs on 10 servers, then the quantity is 10. Similar logic applies to Service items.

See rule AGGR\_SW\_LIC to see how the number of Software licenses is counted. And see Constraint SET\_SOFTWARE\_STPO\_MENGE to see how that value is assigned to the reference characteristic.