DM Community Challenge April 2018
Upsell Rules Corticon
Mike Parish

Rule Statements

Ref	Post	Alias	Text
1	Info	Customer	Bronze or Silver: If they have 1 but not 2 then offer 2, 4 and 5
2	Info	Customer	Bronze or Silver: If they have 1 and 3 but not 6, 7 and 8 then offer 6, 7 and 8
3	Info	Customer	Bronze or Silver: If they have 1 and 2 but not 6, 7 and 8 then offer 4, 5, 7, 8 and 9
4	Info	Customer	Gold: If they have 1 but not 5,6,7 then offer 4,5,7,8,9,10
5	Info	Customer	Platinum: If they have 1 and 2 but not 5,6,7 then offer 4,5 (no cost),7,8 (no cost),9,10

Natural Language View

Conditions	1	2	3	4
What is the customer's combined balance?	[500..2000)	[2000..5000)	[5000..15000)	>= 15000
Actions				
Post Message(s)				
The customer profile is	'Bronze'	'Silver'	'Gold'	'Platinum'

Conditions	0	1	2	3	4	5
What is the customer profile?		{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	'Gold'	'Platinum'
Do they have product 01?		T	T	T	T	T
Do they have product 02?		F	-	T	-	T
Do they have product 03?		-	T	-	-	-
Do they have product 04?		-	-	-	-	-
Do they have product 05?		-	-	-	F	F
Do they have product 06?		-	F	F	F	F
Do they have product 07?		-	F	F	F	F
Do they have product 08?		-	F	F	-	-
Actions						
Post Message(s)		✉	✉	✉	✉	✉
Offer product 1 at this price						
Offer product 2 at this price		95				
Offer product 3 at this price						
Offer product 4 at this price		50		40	30	20
Offer product 5 at this price		50		40	30	0
Offer product 6 at this price			45			
Offer product 7 at this price			45	40	30	20
Offer product 8 at this price			45	40	30	0
Offer product 9 at this price				40	30	20
Offer product 10 at this price					30	20
Comment					'Gold'	'Platinum'

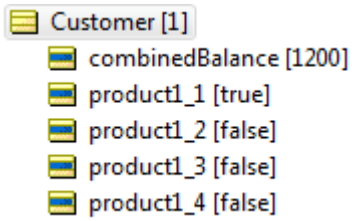
We'll assume that each offer of a new product has an associated cost that is lower for better customers (E.g. Product 4 for Gold and Platinum) and sometimes even waived (E.g. Platinum 5 and 8). If costs were not relevant we could instead use X to indicate the action.

Notice that it may be possible for a customer to get more than one offer of the same product at different prices when they already have 01, 02 and 03 and are Bronze or Silver. So we'll add a rule sheet that determines which is the best deal to offer.

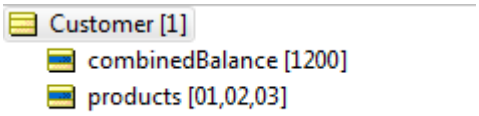
Implementation Approaches

The actual implementation of this logic will depend on how the data objects are defined. In the natural language view we can ignore this since we are just trying to capture the business intent.

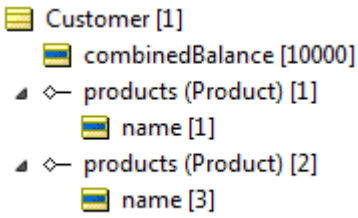
One approach would be to have a Customer with a separate Boolean attribute for each product they might have. Although this is very simple and would work in all decision tables it is not very flexible since a new attribute needs to be added for every new product.



A more flexible approach would be to have a single string that contains a list of all the products the customer has. This allows an arbitrary number of products. This will work in all decision tables that allow substring testing



The most flexible approach is to recognize that products are business objects in their own right and should therefore be associated with the customer as a one-to-many collection. However, not all decision tables support associations. This approach requires a bit more work to set up but it then allows you to have other product specific attributes. This is much harder to implement with the first two data models.



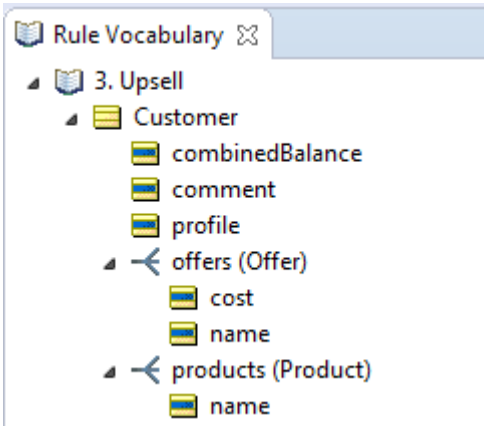
A similar argument applies to the Offers that the rules will create.

An important thing to notice is that the Natural Language form of the rules is independent of which particular method of implementation we choose.

Implementation

The data model (Vocabulary)

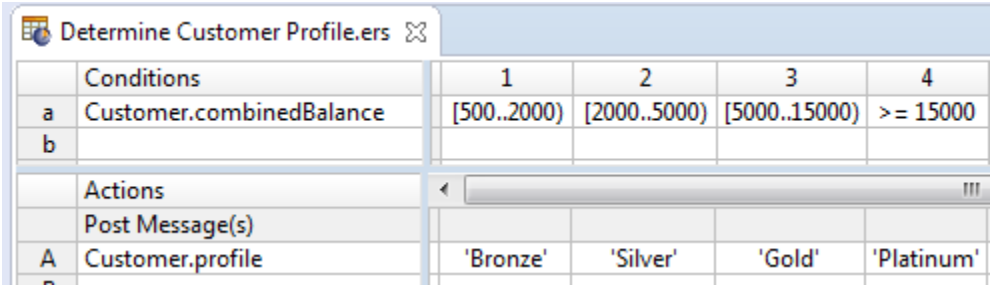
This is the data model we will use for the rest of this discussion. It shows that the Customer object has three attributes (combinedBalance, comment and profile) and two collections – one of products and one of offers.



Structure of the Complete Decision Service

In Corticon the rule flow diagram defines the sequence in which decision tables are to be applied. Effectively it defines the structure of the decision.

Determine Customer Profile



	Conditions	1	2	3	4
a	Customer.combinedBalance	[500..2000)	[2000..5000)	[5000..15000)	>= 15000
b					
	Actions				
	Post Message(s)				
A	Customer.profile	'Bronze'	'Silver'	'Gold'	'Platinum'

This decision table is very simple and just tests the customer combined balance to determine the profile. If it were necessary to calculate the combined balance then we might have an action in the rule sheet something like this Customer.combinedBalance=products.balance->sum

For those ranges that are not explicitly mentioned in the decision table, the profile will be undefined.

In this example that will result in no product offers being made. As a best practice it would be good to include rules dealing with those cases so that a suitable message can be displayed explaining why no offers were forthcoming from the rules.

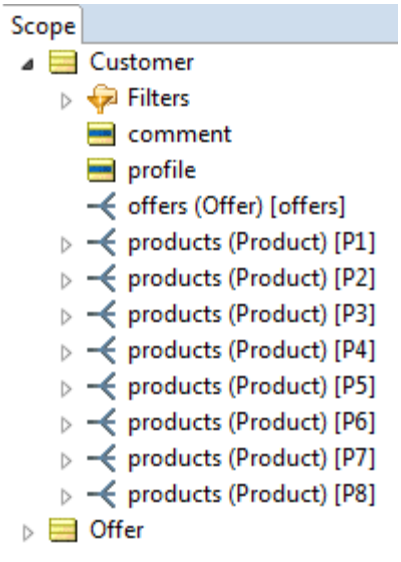
Determine Offers

3. Determine Upsell Products with cost.ers

	Conditions	0	1	2	3	4	5
a	Customer.profile		{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	'Gold'	'Platinum'
b	P1->notEmpty		T	T	T	T	T
c	P2->notEmpty		F	-	T	-	T
d	P3->notEmpty		-	T	-	-	-
e	P4->notEmpty		-	-	-	-	-
f	P5->notEmpty		-	-	-	F	F
g	P6->notEmpty		-	F	F	F	F
h	P7->notEmpty		-	F	F	F	F
i	P8->notEmpty		-	F	F	-	-

	Actions						
	Post Message(s)						
A	offers+=Offer.newUnique[name='01']						
B	offers+=Offer.newUnique[name='02',cost=cellValue]		95				
C	offers+=Offer.newUnique[name='03',cost=cellValue]						
D	offers+=Offer.newUnique[name='04',cost=cellValue]		50		40	30	20
E	offers+=Offer.newUnique[name='05',cost=cellValue]		50		40	30	0
F	offers+=Offer.newUnique[name='06',cost=cellValue]			45			
G	offers+=Offer.newUnique[name='07',cost=cellValue]			45	40	30	20
H	offers+=Offer.newUnique[name='08',cost=cellValue]			45	40	30	0
I	offers+=Offer.newUnique[name='09',cost=cellValue]				40	30	20
J	offers+=Offer.newUnique[name='10',cost=cellValue]					30	20
K	Customer.comment					'Gold'	'Platinum'

The **Scope** section is used to define aliases P1..P8 that reference the particular products in the customer's collection of products.

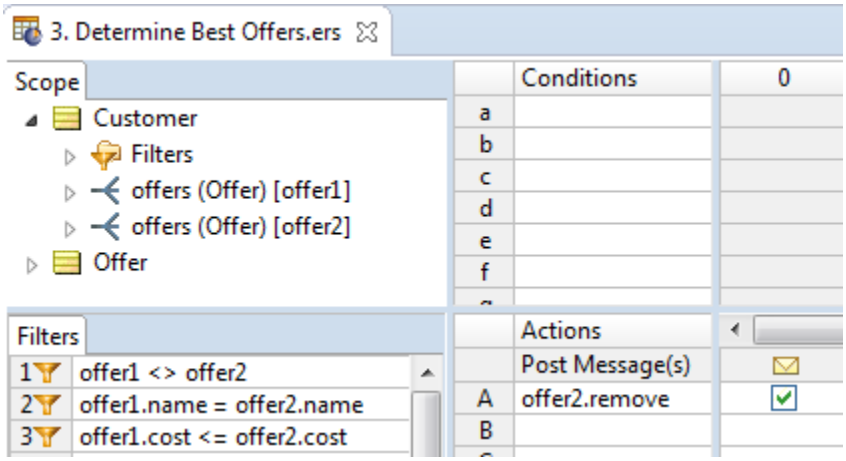


The **Filter** section specifies what is in each alias.

Filters	
1	P1.name='01'
2	P2.name='02'
3	P3.name='03'
4	P4.name='04'
5	P5.name='05'
6	P6.name='06'
7	P7.name='07'
8	P8.name='08'

In the action section of the decision table we create instances of Offers and add them to the customer's collection of offers. Note that at this stage there may be multiple offers for the same product with different prices and so we will need an additional rule sheet that can compare the prices for the same product and pick the lowest price offer for the customer.

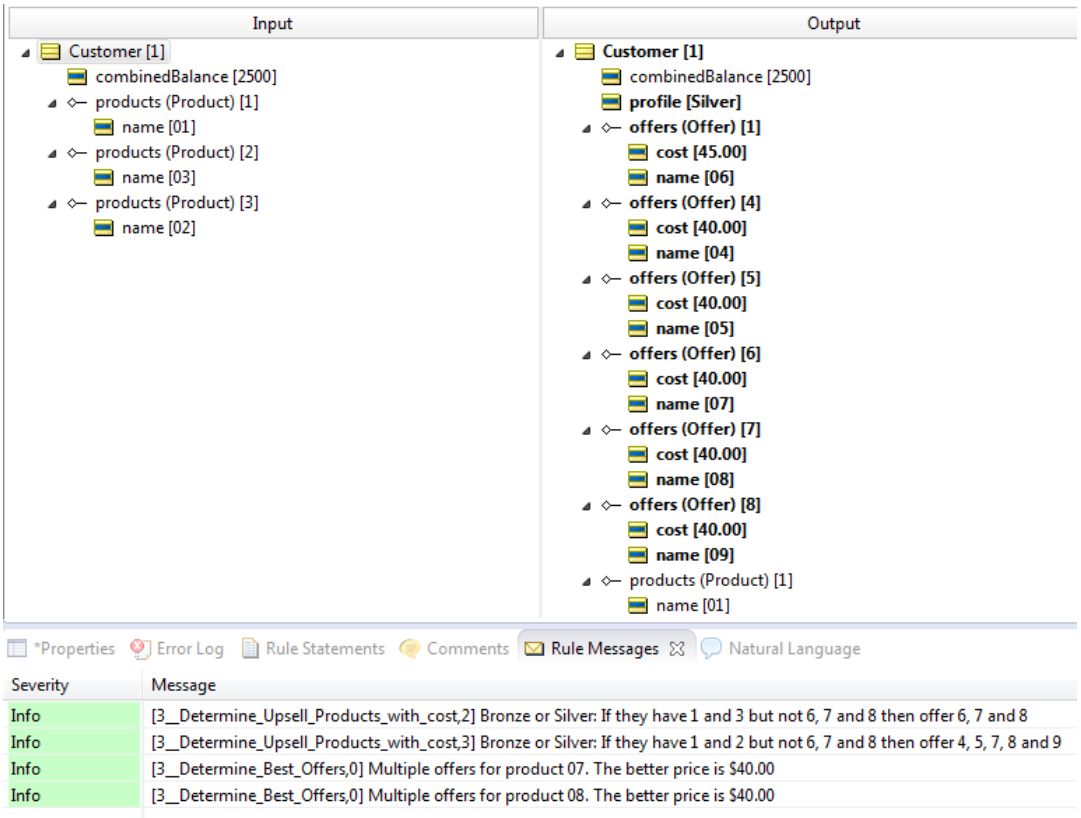
Determine Best Offers



This decision table looks at all pairs of different offers (filter 1) for the same product (filter 2) where one offer has an equal or lower cost (filter 3). The higher cost offer is removed.

Test Case

In this test case, where the customer has products 01, 02 and 03, the customer is offered products 07 and 08 at $45 and $40 by two different rules.
The rules determine that $40 is the better deal in those cases.



Alternative Implementations

The actions will be the same in both cases. Only the conditions need to change.

Products as Separate Attributes of Customer

Each attribute is Boolean, indicating whether the customer has that particular product or not.

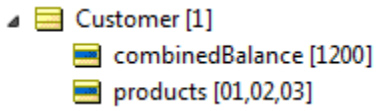
Conditions	0	1	2	3	4	5
Customer.profile		{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	'Gold'	'Platinum'
Customer.product_1		T	T	T	T	T
Customer.product_2		F	-	T	-	T
Customer.product_3		-	T	-	-	-
Customer.product_4		-	-	-	-	-
Customer.product_5		-	-	-	F	F
Customer.product_6		-	F	F	F	F
Customer.product_7		-	F	F	F	F
Customer.product_8		-	F	F	-	-

This gives a very simple decision table that is easy for a business user to maintain but if the products change then the data model in the rules, and the data model in the application that invokes the rules, will need to change.

Products as a Single String Containing a List in Customer

There is only one attribute for the products, of type string, but it needs to be formatted as a list.

For example:



Now the application that calls the rules does not need to change in order to add new products. The rules of course will need to be changed. One requirement for this approach is that the product names must not be substrings of other product names. Otherwise you'd get false matches. E.g. 1 and 10

Conditions	0	1	2	3	4	5
Customer.profile		{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	{'Bronze', 'Silver'}	'Gold'	'Platinum'
Customer.products.contains('01')		T	T	T	T	T
Customer.products.contains('02')		F	-	T	-	T
Customer.products.contains('03')		-	T	-	-	-
Customer.products.contains('04')		-	-	-	-	-
Customer.products.contains('05')		-	-	-	F	F
Customer.products.contains('06')		-	F	F	F	F
Customer.products.contains('07')		-	F	F	F	F
Customer.products.contains('08')		-	F	F	-	-