Duplicates – A Corticon Approach (no java needed)– Mike Parish 8/3/2015

The Problem

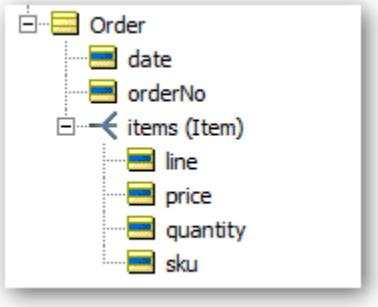
Let's assume that you have a sales order like the one below:

Product SKU	Price	Quantity
SKU-1000	10	10
SKU-1	20	3
SKU-2	30	1
SKU-1	20	4
SKU-3	40	6
SKU-3	42	8
SKU-4	15	2

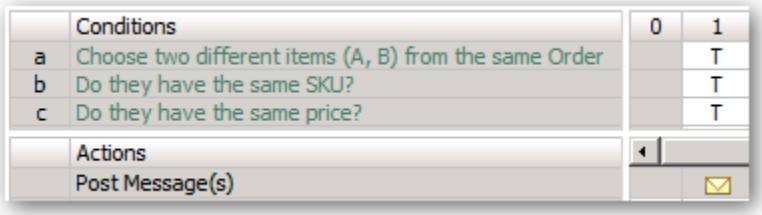
Create a decision model that is capable of deciding which product lines inside such orders are duplicated. The duplicate product lines can be recognized by user-defined similarity rules, e.g. two product lines are considered "duplicate" if they have the same "sku" and "price".
P.S. This problem was initially proposed by Antonio Plais in this LinkedIn discussion. Different approaches to the problem from the DMN perspective can be found here.

Solution

Vocabulary



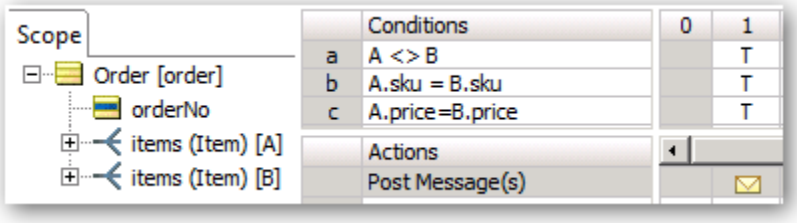
Natural Language View



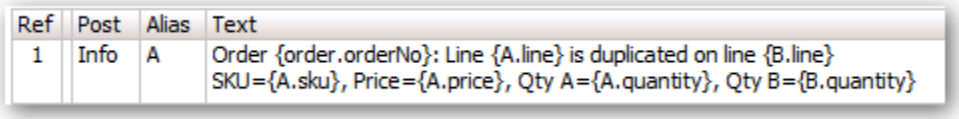
The little envelope icon indicates that a message should be displayed.

Implementation View

The Scope section is where you specify that A and B are two items in the same Order.
Condition (a) specifies that A and B must be different items (though they may have identical values)



The message is defined as:

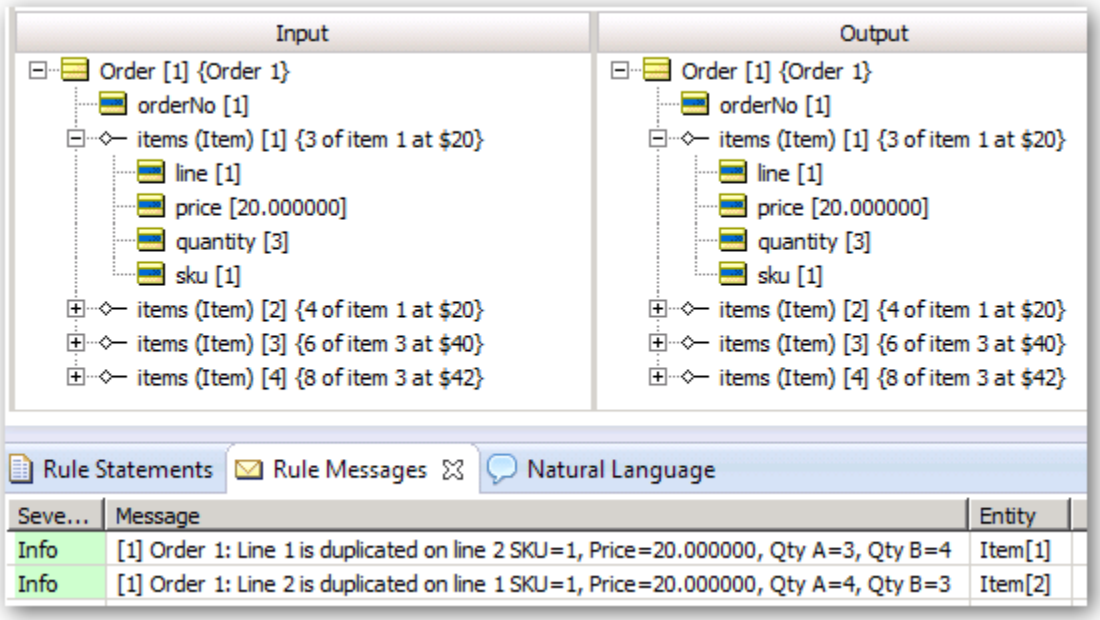


Ref	Post	Alias	Text
1	Info	A	Order {order.orderNo}: Line {A.line} is duplicated on line {B.line} SKU={A.sku}, Price={A.price}, Qty A={A.quantity}, Qty B={B.quantity}

The entries in {} will have actual values substituted when the rules execute

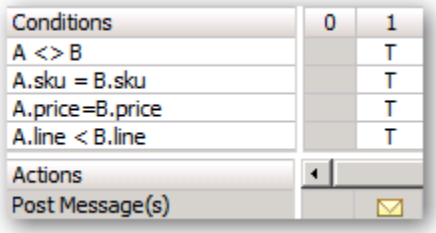
This is all that is required in Corticon! There is nothing hidden and no special "duplicate" function written in java (which would require a programmer every time we need to modify our rules for "duplicate")

Test Case

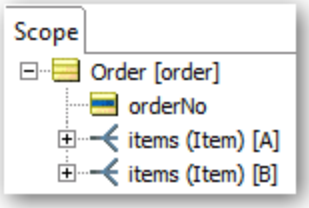


Notice that two messages are displayed. That's because line 1 is duplicated on line 2 and (obviously) line 2 is duplicated on line 1. Corticon naturally (and blindly) considers all possible combinations of two items in any order (unless we tell it to do something differently)
If there were more duplicates the number of messages could start to get large. Just 4 identical items would produce 12 messages (4 * 3).

In order to display just one message per duplicate we could do it by imposing some sequencing on the items. We'll do it here by displaying the message only for the smaller line number.
Like this:



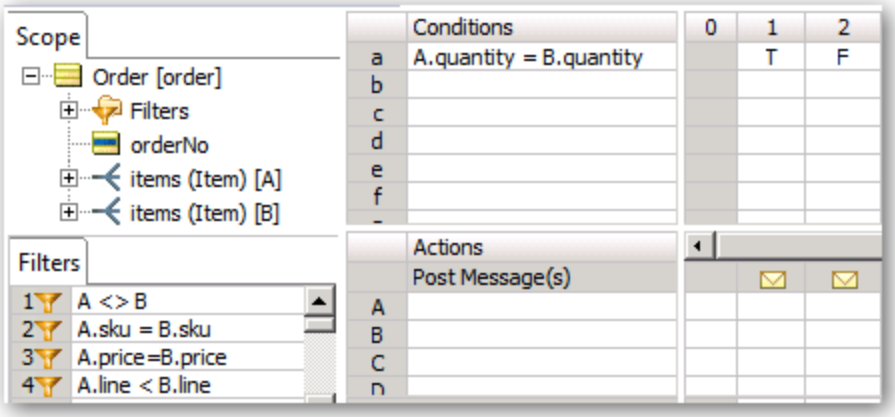
Corticon is able to solve this easily because of its "Scope" section where we show the context of the decision.
In this case the decision applies to a single Order (if there are many then Corticon will treat each separately). We also define two "aliases" called A and B. The natural behavior of Corticon is to allow A and B to independently bind to any of the items in an order. Then in the rule conditions Corticon will try every possible combination of two items in the order.



The reason we include A<>B is to tell Corticon not to compare an item with itself.
The rules then simply define the conditions for a duplicate.

Using Databases and Filters

Suppose the order data is in a database. We wouldn't want to read millions of orders into memory just to find the few with duplicates.
Filters can be used in Corticon to set up "entry requirements" for data to get into the rule sheet.



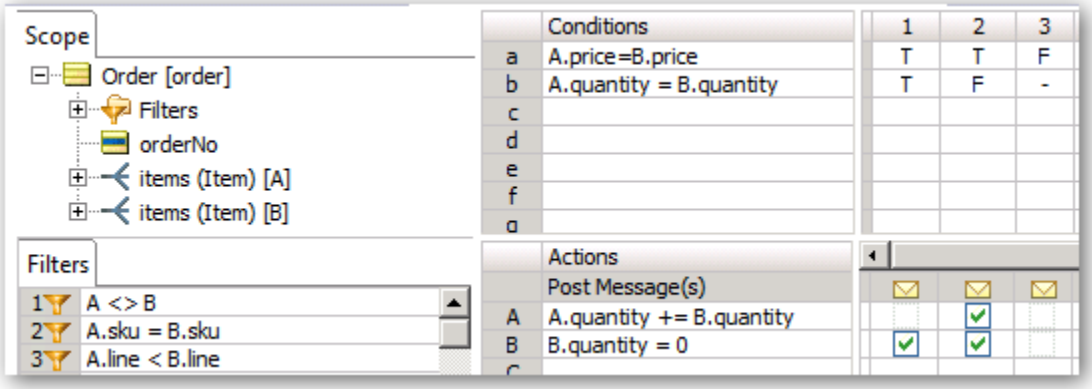
By moving some of the conditions to the filter section and marking the filters as "Database Filter" Corticon will construct an appropriate SQL query to retrieve records that meet the filter requirement and only load these into memory for processing by the rule sheet.

Ref	Post	Alias	Text
1	Info	A	Exact duplicate - same sku, price and quantity
2	Info	A	Close duplicate - same sku, price but different quantity

Only items A and B that meet the filter criteria will be processed by the rules in columns 1 and 2. In this case we can display a suitable message when the quantities are the same or different.

Real World Scenario

Although it's easy to identify the duplicates with a message, it's more likely that we'd want the rules to take some action to resolve the duplicate. For example we might decide that for exact duplicates (sku, price, qty) we'll eliminate the duplicate item. But for duplicates with different quantities we'll merge the two items into one by summing the two quantities. And we might also want rules to deal with the situation where the same sku on an order has two different prices – maybe it's an error. Or maybe the price reflects quantity discounts that have been applied and we need to consolidate the two items and revise the price. Or maybe we give the customer the lower of the two prices.
A further refinement of the rule sheet might look like this where some of the work is done in the filter some in the rule conditions and some in the actions:



Basically conditions that are the same across all rules can be defined once in the filter leaving only the variations across the rule columns.
With messages as follows:

Ref	Post	Alias	Text
1	Info	A	Exact duplicate - same sku, price and quantity - delete one of them
2	Info	A	Close duplicate - same sku, price but different quantity - Merge them
3	Warning	A	Duplicates with different pricing for same SKU

Summary

Dealing with collections of data and associations between these collections is very common in virtually all non-trivial business rules applications. Corticon manages this by using

1. The Scope to define the context and relationships between objects
2. The Filters to define what data is relevant to the rules in a particular rule sheet

By using these features, Corticon is easily able to solve the "Duplicate" problem without resorting to coding a special function in java. This gives the business user complete control and flexibility in defining what is meant by "Duplicates" and what actions to take in the event they occur.

It's the Scope and Filter sections that distinguish the Corticon decision table from all others and make it possible to solve even very complex problems.