Santa’s Reindeer - Alternative Approaches in Corticon – Mike Parish

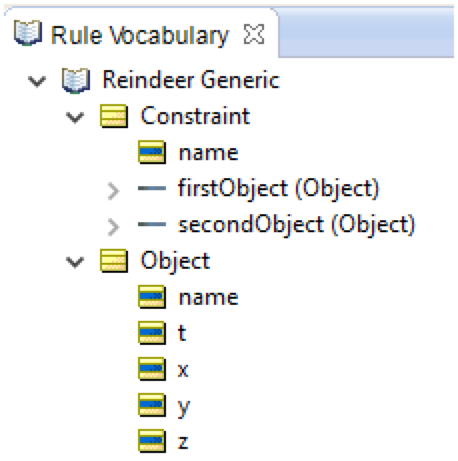
Two additional approaches to solving the Reindeer problem in Corticon are presented below.
The first shows how decision tables can be used to generalize the solution to any kind of object in n-dimensional space.
The second shows how decision tables can be used for constraint based solving (i.e. trying all possibilities)

Generalizing the Decision Table Solution

Bob Moore posed an interesting question about how to model this problem in a generic fashion where the data and rules are passed in rather than hard coded in the decision tables.
He asked “If I don’t have the reindeer names and the rules when I’m designing my decision table, what do I use for the rules/rows? For that matter what do I use for attributes/columns? “

Fortunately there is a solution to this (at least in Corticon and possibly one or two other rule engines).

We can define a generic vocabulary such as this using the concept of associations.



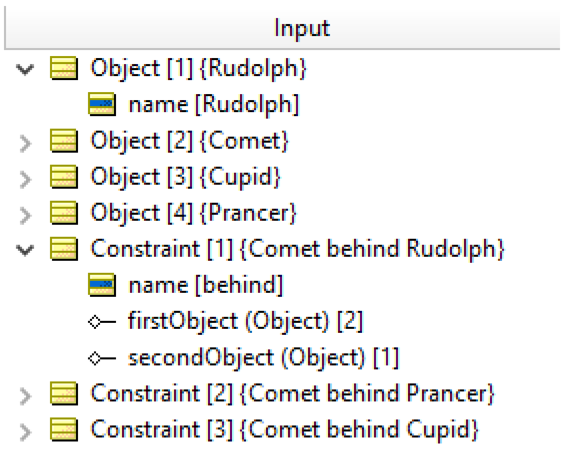
There are two kinds of entity.
The first defines the constraints on the objects by giving the name of the constraint and the two objects to be constrained by it.
The second simply defines the objects (Reindeer, boxes, whatever).

The rule sheet itself defines what is meant by “behind” – in this case “behind” means having a higher x coordinate.

Note that this model doesn't care if it's reindeer, or people or boxes of cookies.

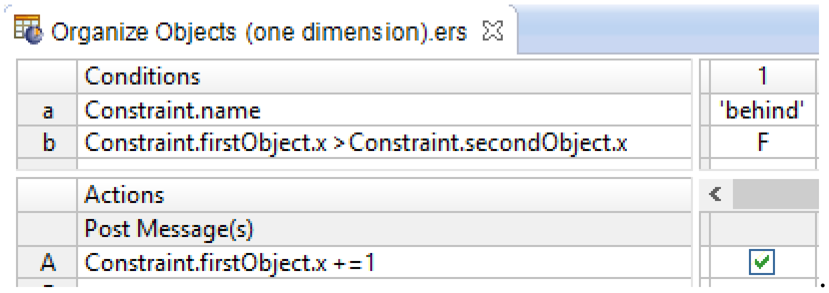
It also provides for ordering in three dimensions by storing the x, y, z coordinates so we can support the notions of above and below, and left and right in addition to in front and behind (and we could easily extend this n dimensional space).

Here's an example of the data that might be sent to the rule engine specifying the objects and the required constraints.



```
Input
Object [1] {Rudolph}
    name [Rudolph]
Object [2] {Comet}
Object [3] {Cupid}
Object [4] {Prancer}
Constraint [1] {Comet behind Rudolph}
    name [behind]
    firstObject (Object) [2]
    secondObject (Object) [1]
Constraint [2] {Comet behind Prancer}
Constraint [3] {Comet behind Cupid}
```

Now we can define an entirely generic decision table using the same basic pattern as the first Corticon solution (increasing the coordinates of objects that break the constraints):



Organize Objects (one dimension).ers

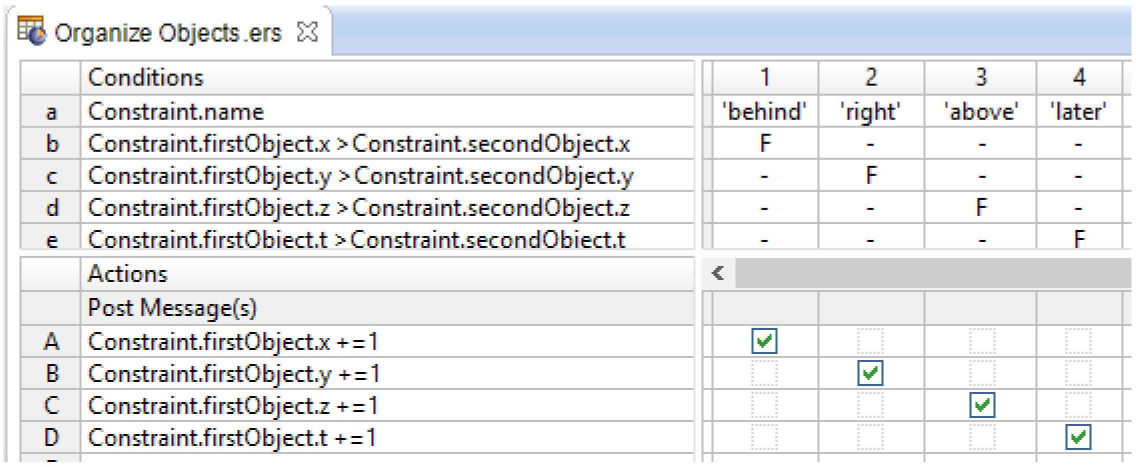
	Conditions	1
a	Constraint.name	'behind'
b	Constraint.firstObject.x > Constraint.secondObject.x	F

	Actions	
	Post Message(s)	
A	Constraint.firstObject.x += 1	✓

So instead of 15 rules in the decision table there is just one. Even if there were 1000 constraint rules this one generic rule would take care of them all.

The 15 rules will, of course, need to be represented as instances of the Constraint entity in the input data. And, thanks to Bob's Python, we could automatically generate this from the initial rule statements (Corticon could also do the same parsing using it's built in string functions). There will be 30 such constraints, though as Bob points out, far fewer are actually needed. Reducing a set of constraints to the minimum set would be an interesting challenge problem.

This basic pattern can then easily be extended to deal with the y and z (and more) coordinates



Organize Objects.ers

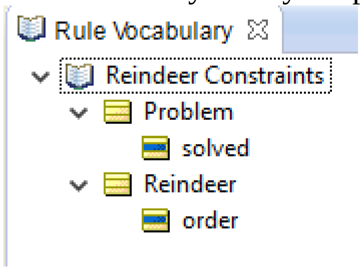
	Conditions	1	2	3	4
a	Constraint.name	'behind'	'right'	'above'	'later'
b	Constraint.firstObject.x > Constraint.secondObject.x	F	-	-	-
c	Constraint.firstObject.y > Constraint.secondObject.y	-	F	-	-
d	Constraint.firstObject.z > Constraint.secondObject.z	-	-	F	-
e	Constraint.firstObject.t > Constraint.secondObject.t	-	-	-	F
	Actions				
	Post Message(s)				
A	Constraint.firstObject.x +=1	✓			
B	Constraint.firstObject.y +=1		✓		
C	Constraint.firstObject.z +=1			✓	
D	Constraint.firstObject.t +=1				✓

So instead of putting reindeer in a single line we might be storing boxes of cookies on shelves in a warehouse by aisle, column and shelf number.

Constraint Approach

For completeness here is a constraint based approach modeled in Corticon. It's very similar to the OpenRules approach.

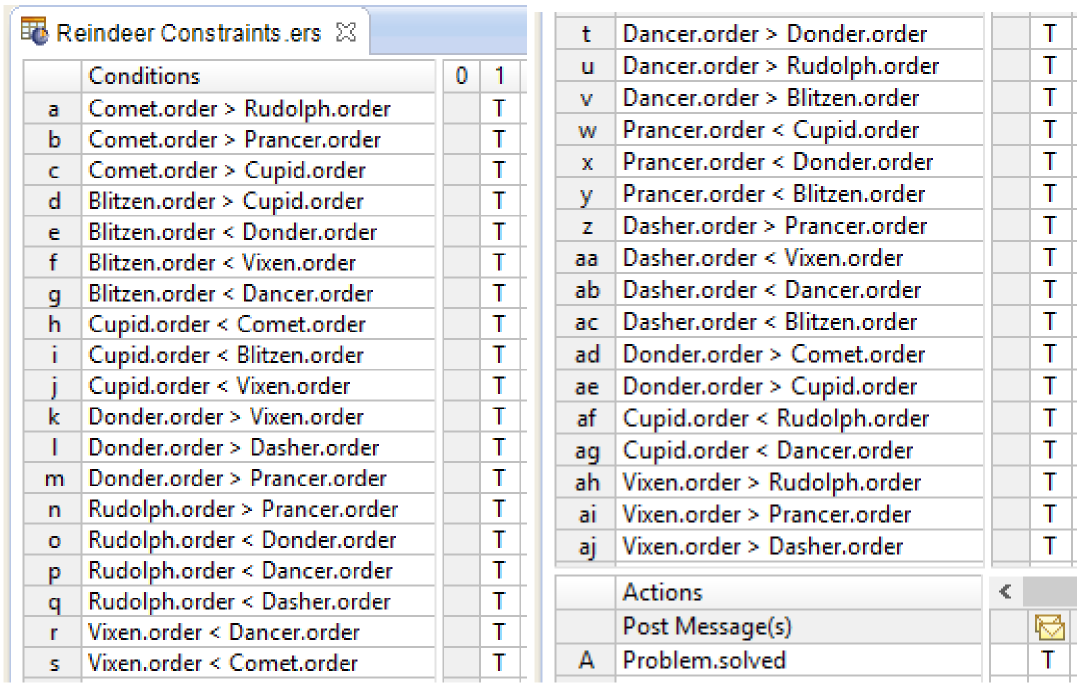
The vocabulary is very simple:



Rule Vocabulary

- Reindeer Constraints
  - Problem
    - solved
  - Reindeer
    - order

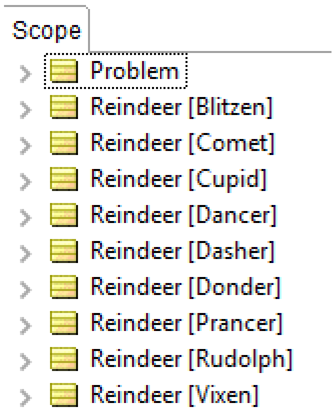
And there is just one rule sheet that expresses the constraints:



Reindeer Constraints.ers

	Conditions	0	1
a	Comet.order > Rudolph.order		T
b	Comet.order > Prancer.order		T
c	Comet.order > Cupid.order		T
d	Blitzen.order > Cupid.order		T
e	Blitzen.order < Donder.order		T
f	Blitzen.order < Vixen.order		T
g	Blitzen.order < Dancer.order		T
h	Cupid.order < Comet.order		T
i	Cupid.order < Blitzen.order		T
j	Cupid.order < Vixen.order		T
k	Donder.order > Vixen.order		T
l	Donder.order > Dasher.order		T
m	Donder.order > Prancer.order		T
n	Rudolph.order > Prancer.order		T
o	Rudolph.order < Donder.order		T
p	Rudolph.order < Dancer.order		T
q	Rudolph.order < Dasher.order		T
r	Vixen.order < Dancer.order		T
s	Vixen.order < Comet.order		T
t	Dancer.order > Donder.order		T
u	Dancer.order > Rudolph.order		T
v	Dancer.order > Blitzen.order		T
w	Prancer.order < Cupid.order		T
x	Prancer.order < Donder.order		T
y	Prancer.order < Blitzen.order		T
z	Dasher.order > Prancer.order		T
aa	Dasher.order < Vixen.order		T
ab	Dasher.order < Dancer.order		T
ac	Dasher.order < Blitzen.order		T
ad	Donder.order > Comet.order		T
ae	Donder.order > Cupid.order		T
af	Cupid.order < Rudolph.order		T
ag	Cupid.order < Dancer.order		T
ah	Vixen.order > Rudolph.order		T
ai	Vixen.order > Prancer.order		T
aj	Vixen.order > Dasher.order		T

	Actions		
	Post Message(s)		✉
A	Problem.solved		T



Scope

- Problem
- Reindeer [Blitzen]
- Reindeer [Comet]
- Reindeer [Cupid]
- Reindeer [Dancer]
- Reindeer [Dasher]
- Reindeer [Donder]
- Reindeer [Prancer]
- Reindeer [Rudolph]
- Reindeer [Vixen]

The reindeer names in the conditions are aliases to the collection of possible positions.

When supplied with the nine positions for a reindeer (1…9) Corticon automatically tries every combination until all constraints are satisfied.
With just nine reindeer Corticon is almost instantaneous.

1000 reindeer would be more challenging for this approach – but very easy for the production rule approach discussed in a separate Corticon solution.