Decision Management Community Challenge

July 2022

Seth Meldon

Progress Corticon

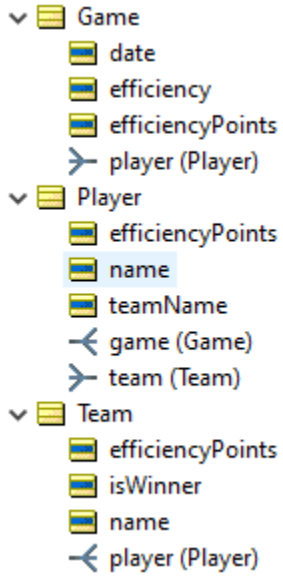
Prompt:

Team	Player	Game Date	Efficiency
Mustungs	Brown	4/1/2022	good
	Brown	4/2/2022	better
	Brown	4/3/2022	best
	Robinson	4/1/2022	worst
	Robinson	4/2/2022	better
	Robinson	4/3/2022	best
	Smith	4/1/2022	bad
	Smith	4/2/2022	good
	Smith	4/3/2022	bad
Eagles	Black	4/1/2022	good
	Black	4/2/2022	better
	Black	4/3/2022	best
	White	4/1/2022	worst
	White	4/2/2022	better
	White	4/3/2022	best
	Green	4/1/2022	bad
	Green	4/2/2022	good
	Green	4/3/2022	worst

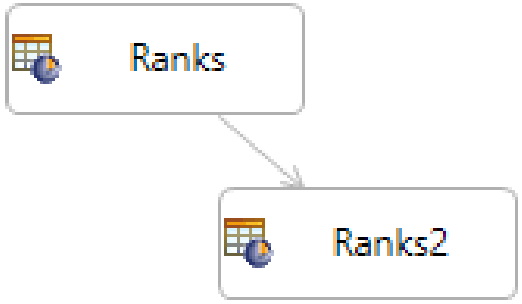
Your decision model should evaluate performance of different teams based on efficiency of their players. On the left you can see how different players performed during different games. Each player receives 5 points for “best” efficiency, 3 points for “better” efficiency, 2 for “good” efficiency. You need to subtract 2 points for “bad” efficiency and 5 points for “worst” efficiency. Which team got the most points?

Please submit your solutions using your favorite BR/DM tools.

First we'll define a data model to represent the various data fields, this will be defined in Corticon **Rule Vocabulary**:

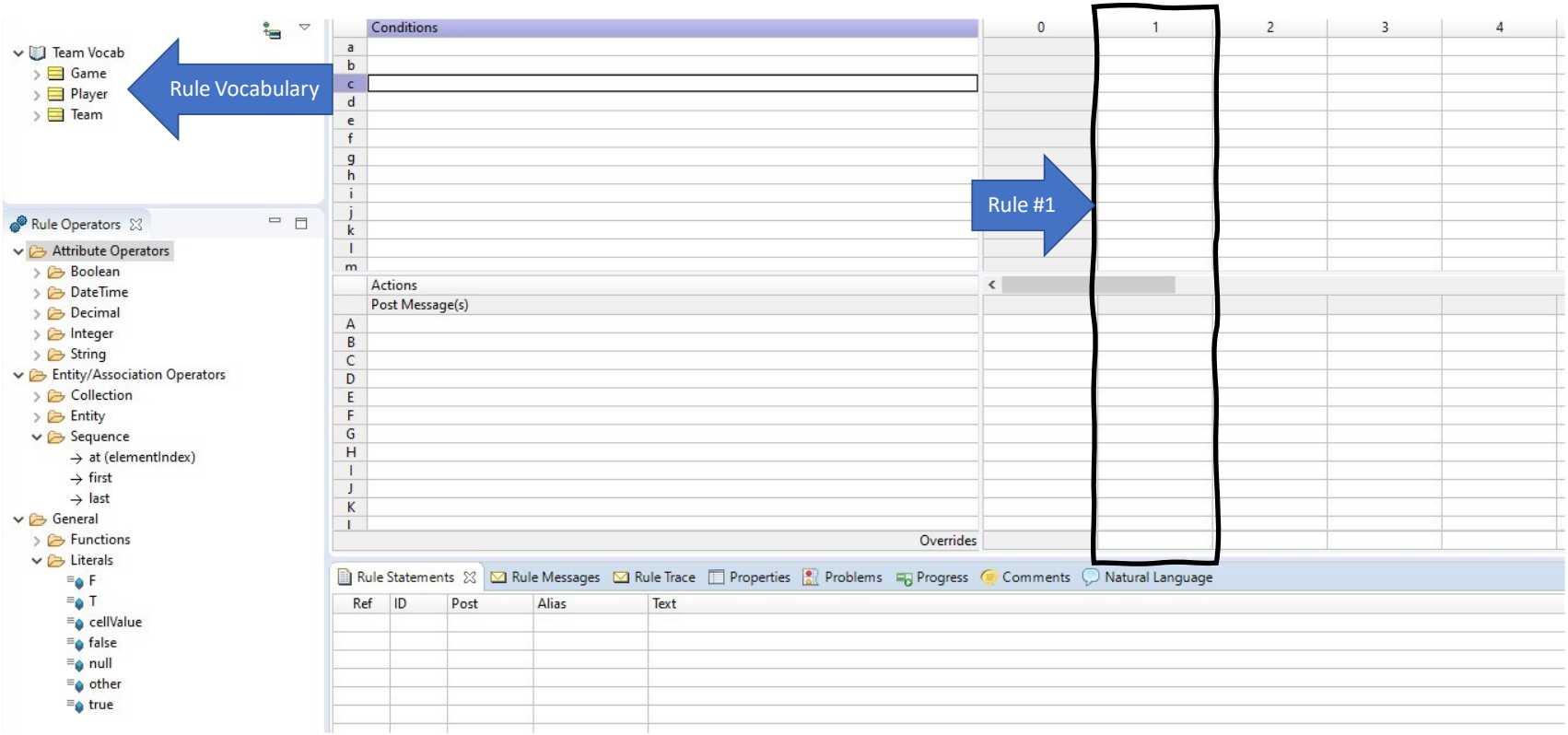


The decision service in compiled from a **Ruleflow**, which contains two **Rulesheets**:

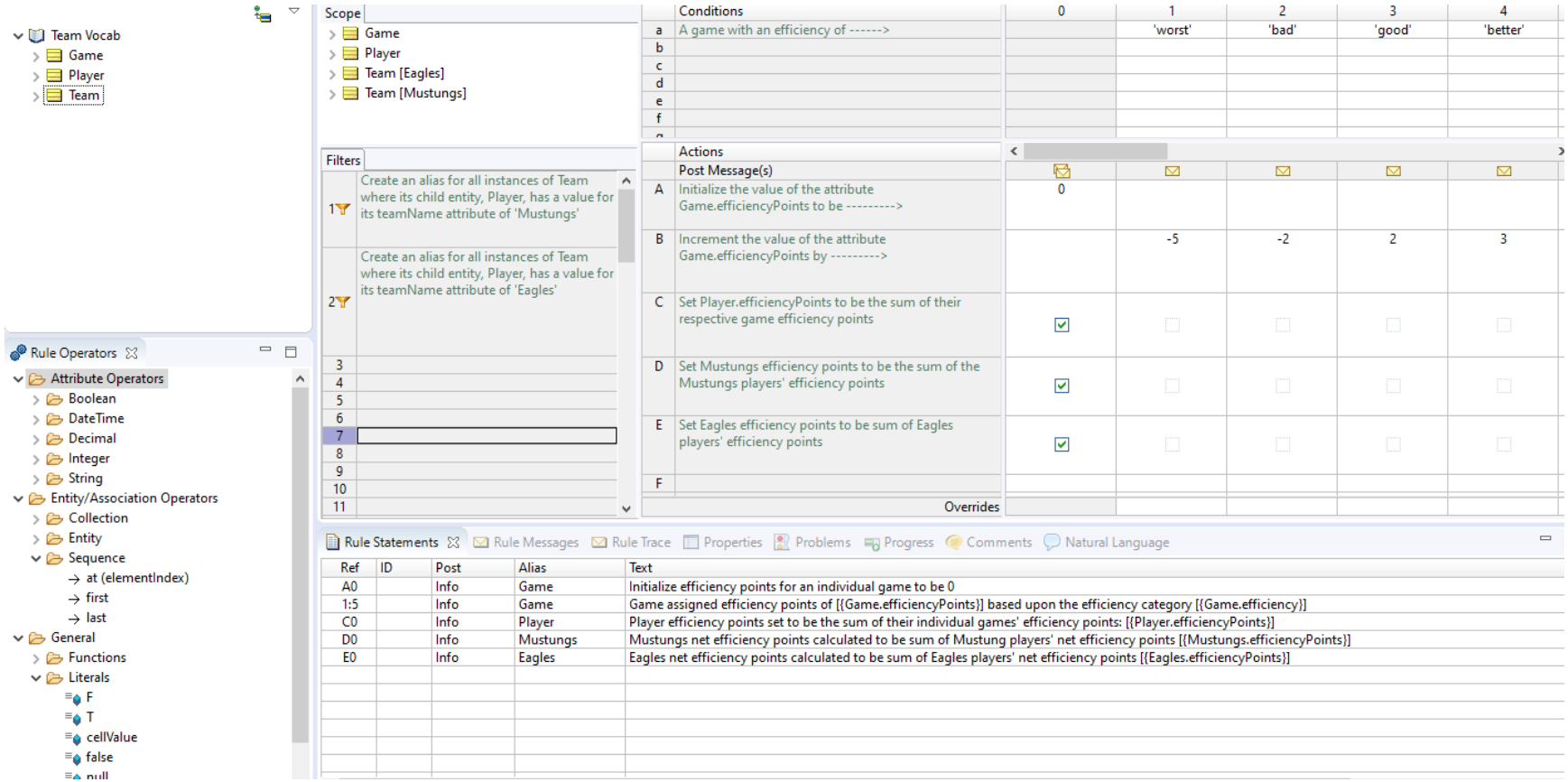


Rules are divided into Conditions and Actions in a rulesheet. Each column is a discrete rule contains any number of conditions (e.g. when Game.efficiency = 'Good') that may produce any number of actions (continuing the same example, we assign the value of 2 to the field Game.efficiencyPoints). Column 0 is an 'action only' rule, so it will always fire.

Blank rulesheet:

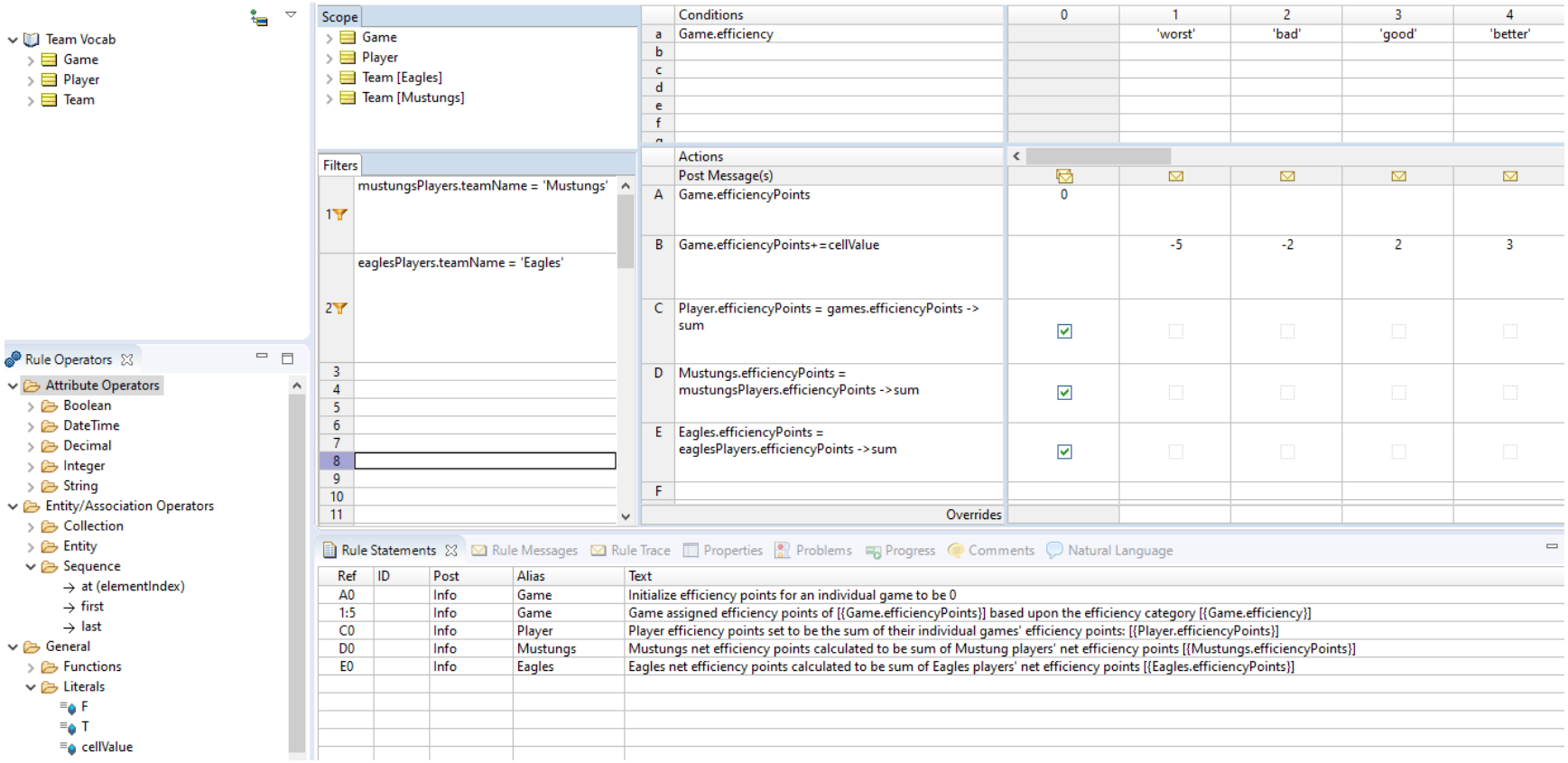


For both documentation and to simplify new user onboarding, we optionally can define Natural Language values to overlay over the completed syntax in a rulesheet:

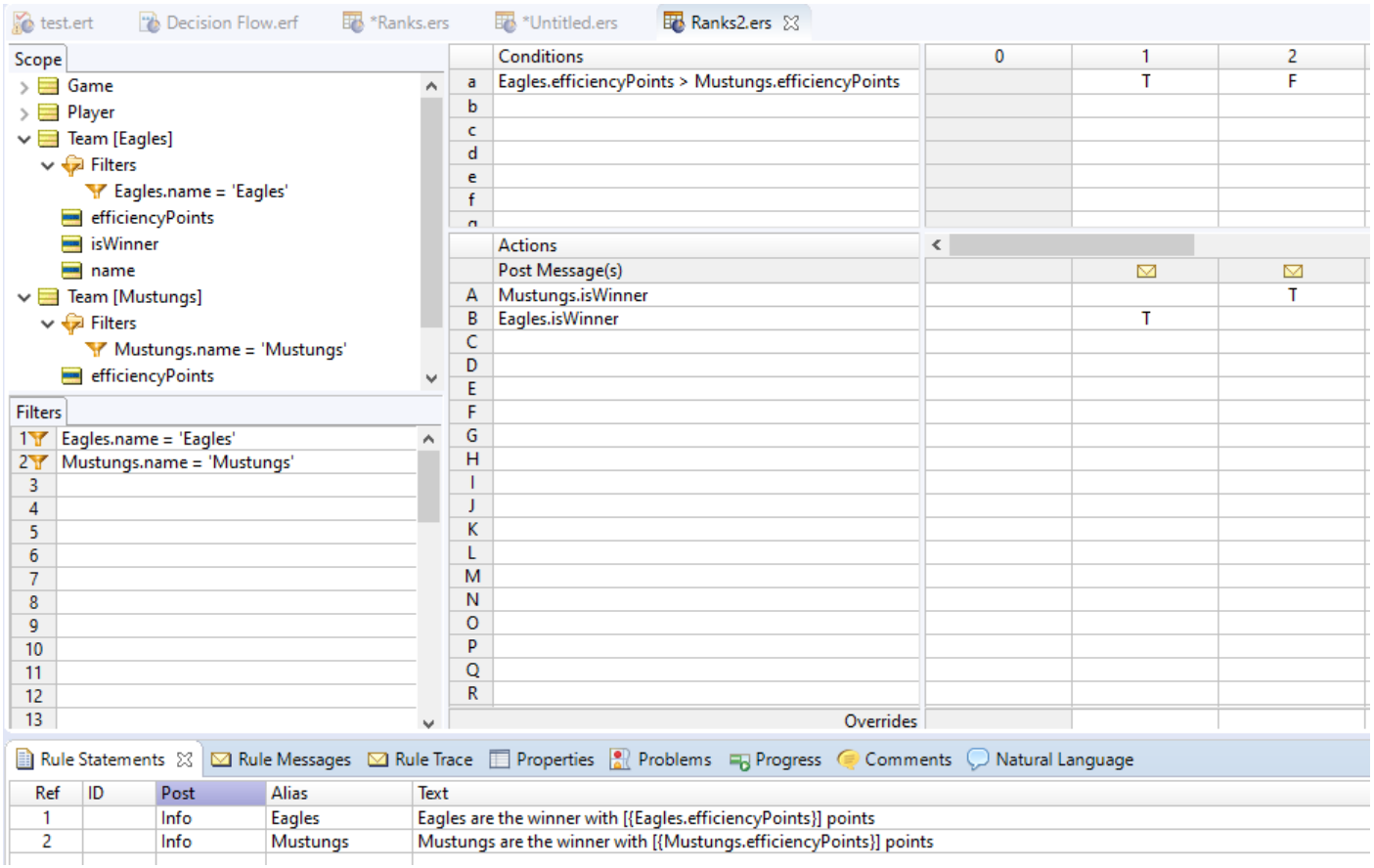


These overlays are different Rule Statements. Rule Statements, while optional, are quite useful for rule observability. They will be sent back to the invoking application for all of the rules that fire—and only the rules that fire. This provides a clear description of each rule that is executed.

Below is the first rulesheet (the same one as the above) with Natural Language view toggled off. To best understand the function of the different aspects defined in this rulesheet, revisit the natural language descriptions in the rulesheet above.

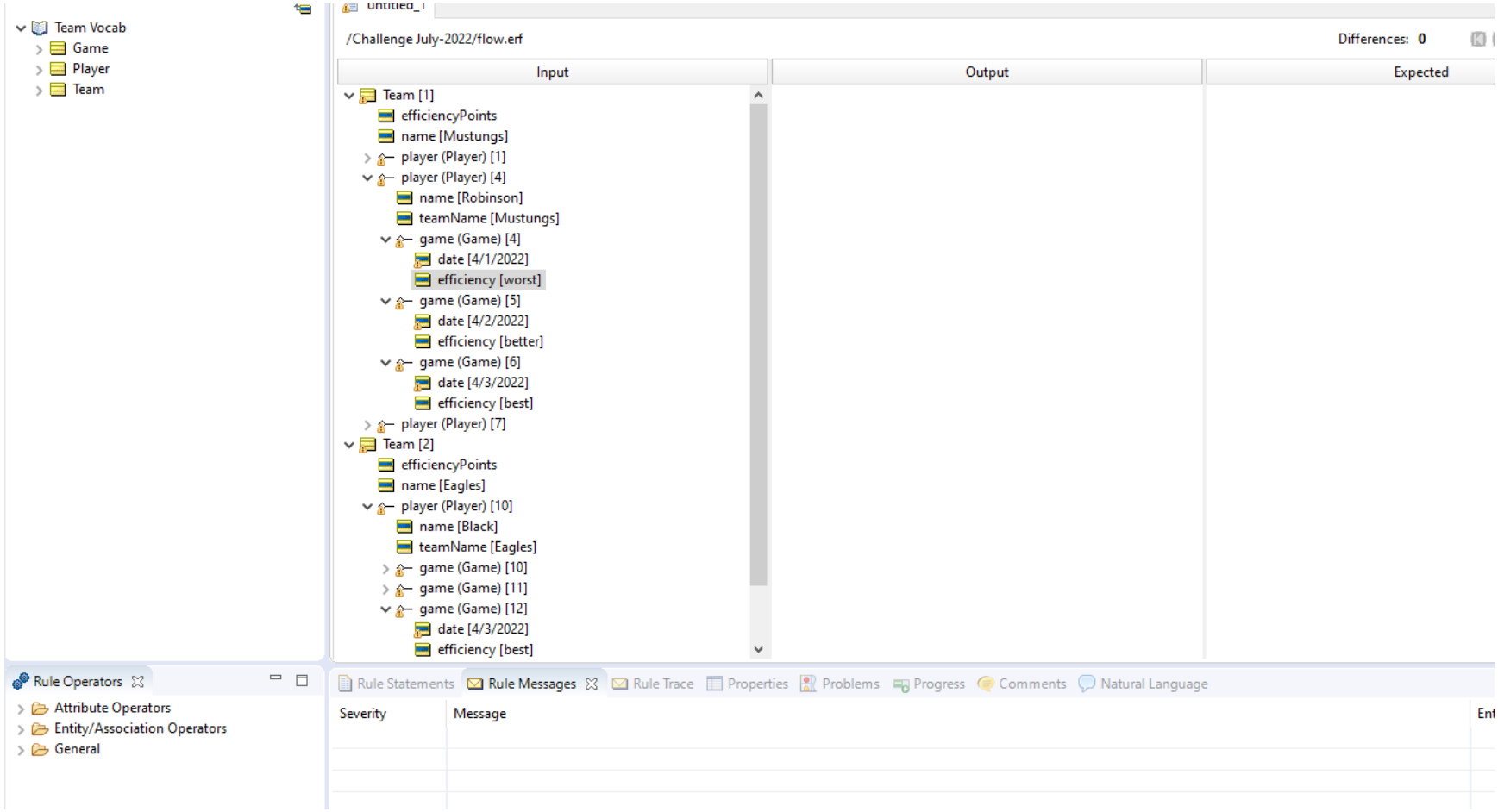


The second rulesheet is simpler, but also makes use of a concept called Filtered Aliases to represent how Corticon should handle different scenarios involving the same entity/attributes. We can thus define rule rules applicable only to the entity ‘Team’ only where its underlying ‘teamName’ attribute equals ‘Eagles’ and others for where it equals ‘Mustungs’.

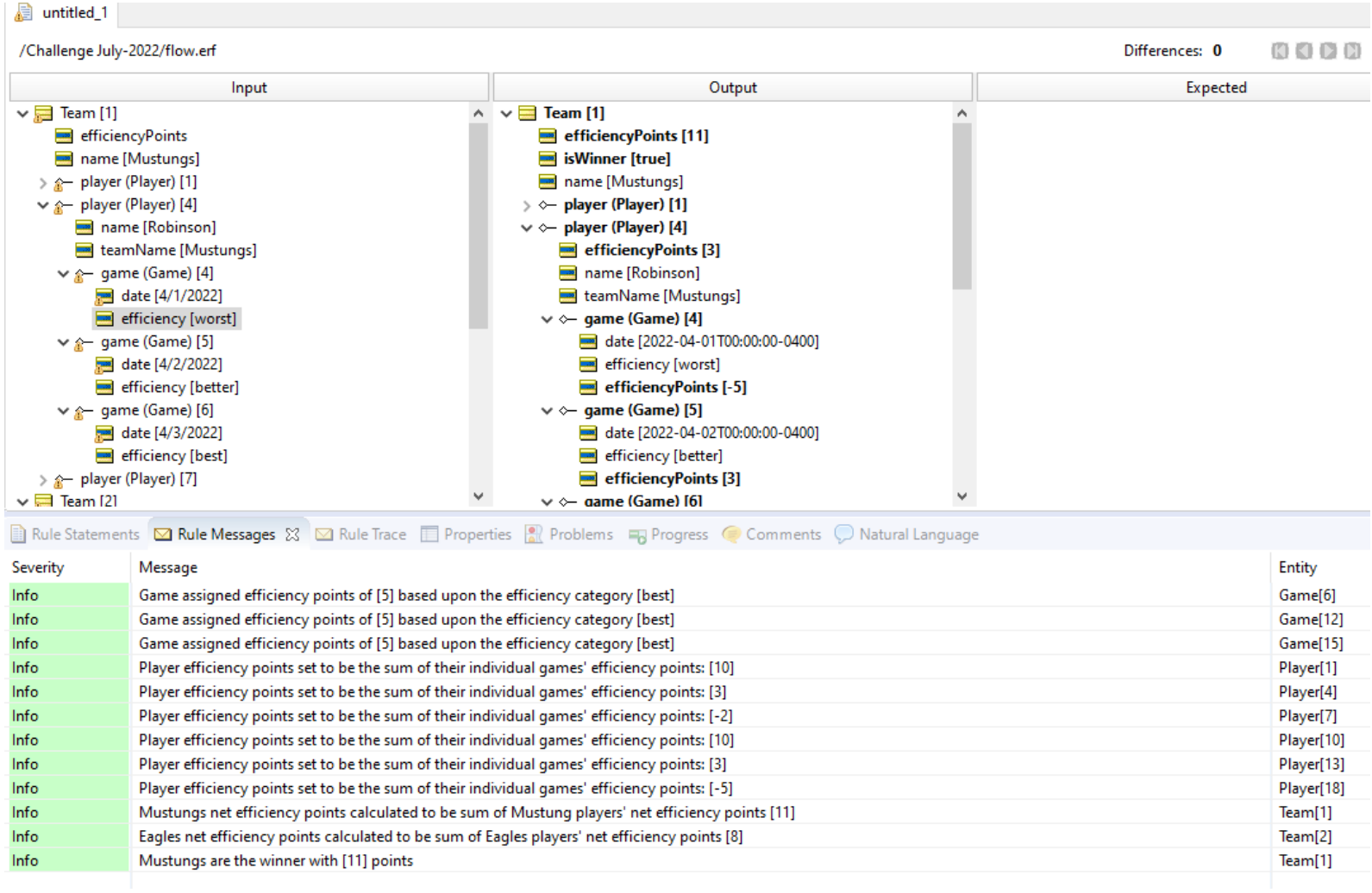


Prior to deploying the rules as a service we can define a rule test to validate the outputs produced by the rules for test inputs:

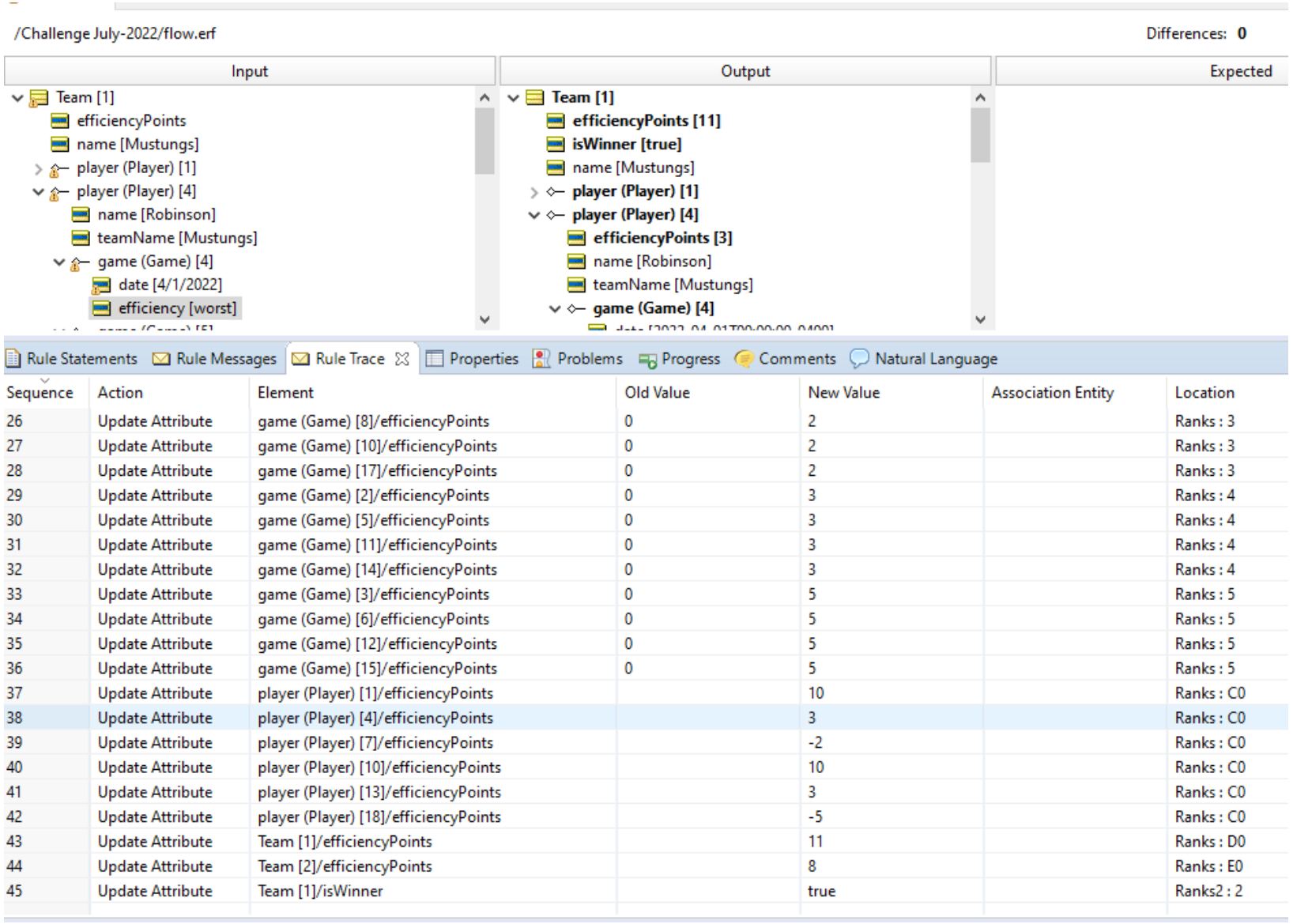
Prior to running test:



After running test:

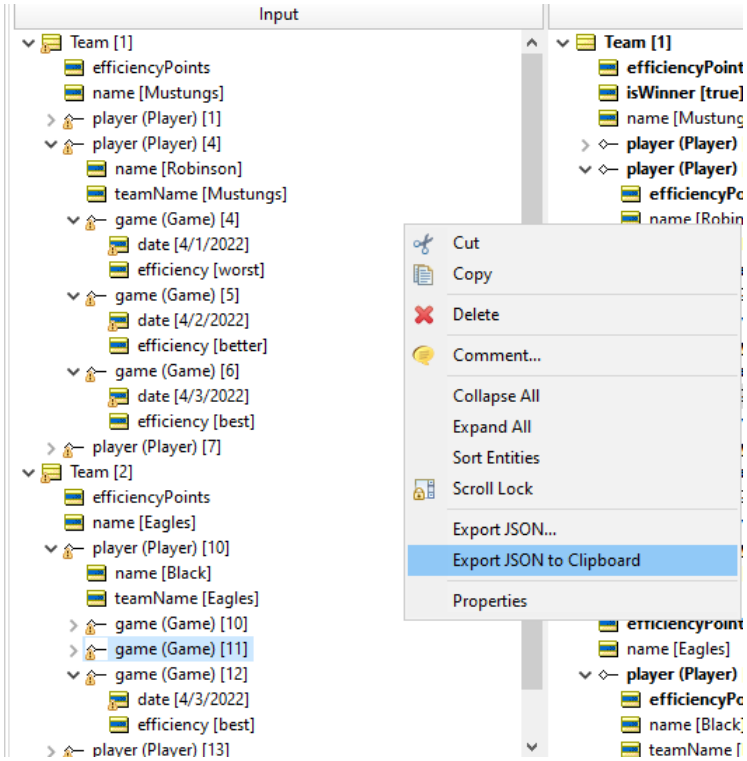


If we toggle to the ‘Rule Trace’ panel, we can see precisely what changes the rules made to the input payload, the previous/new values, and the chronology of these changes:

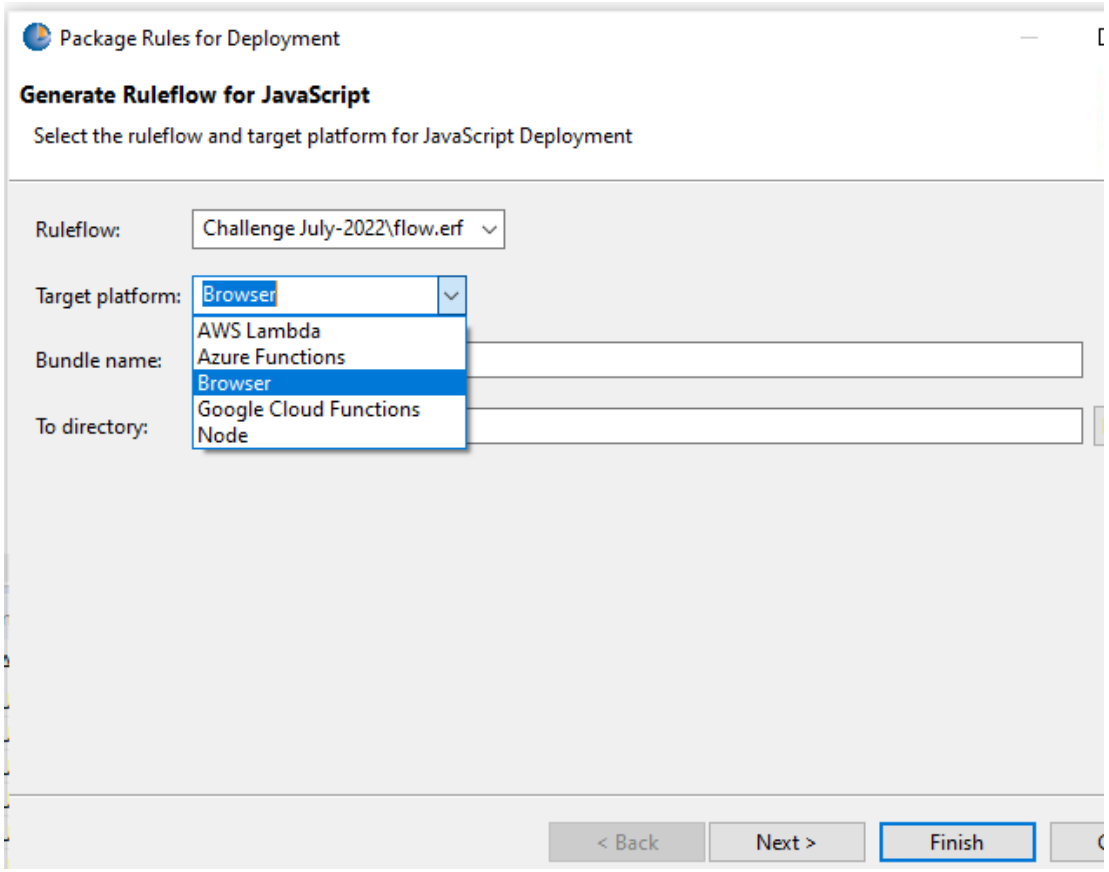


Sequence	Action	Element	Old Value	New Value	Association Entity	Location
26	Update Attribute	game (Game) [8]/efficiencyPoints	0	2		Ranks : 3
27	Update Attribute	game (Game) [10]/efficiencyPoints	0	2		Ranks : 3
28	Update Attribute	game (Game) [17]/efficiencyPoints	0	2		Ranks : 3
29	Update Attribute	game (Game) [2]/efficiencyPoints	0	3		Ranks : 4
30	Update Attribute	game (Game) [5]/efficiencyPoints	0	3		Ranks : 4
31	Update Attribute	game (Game) [11]/efficiencyPoints	0	3		Ranks : 4
32	Update Attribute	game (Game) [14]/efficiencyPoints	0	3		Ranks : 4
33	Update Attribute	game (Game) [3]/efficiencyPoints	0	5		Ranks : 5
34	Update Attribute	game (Game) [6]/efficiencyPoints	0	5		Ranks : 5
35	Update Attribute	game (Game) [12]/efficiencyPoints	0	5		Ranks : 5
36	Update Attribute	game (Game) [15]/efficiencyPoints	0	5		Ranks : 5
37	Update Attribute	player (Player) [1]/efficiencyPoints		10		Ranks : C0
38	Update Attribute	player (Player) [4]/efficiencyPoints		3		Ranks : C0
39	Update Attribute	player (Player) [7]/efficiencyPoints		-2		Ranks : C0
40	Update Attribute	player (Player) [10]/efficiencyPoints		10		Ranks : C0
41	Update Attribute	player (Player) [13]/efficiencyPoints		3		Ranks : C0
42	Update Attribute	player (Player) [18]/efficiencyPoints		-5		Ranks : C0
43	Update Attribute	Team [1]/efficiencyPoints		11		Ranks : D0
44	Update Attribute	Team [2]/efficiencyPoints		8		Ranks : E0
45	Update Attribute	Team [1]/isWinner		true		Ranks2 : 2

To test the JavaScript bundle we create, we can export the JSON from the input to the clipboard. This will be revisited momentarily.



Finally, we can generate JavaScript bundle which will run directly within a browser so that you can test the performance of the logic for yourself.



The output is a decisionServiceBundle.js file and a sample HTML page invoking the JavaScript based upon test data. These can be accessed online using Codepen here. The page's 'input' side has been prepopulated with the JSON copied to the clipboard, which corresponds with the data from the player performance spreadsheet provided in the prompt.