Declarative Decision Modeling with Rule Solver

Integrating Rule Engine and Constraint Solver
To support Declarative Decision Modeling

OPEN RULES DECISION MANAGER + OPEN RULES RULE SOLVER

Jacob Feldman, PhD
OpenRules. Inc., CTO
www.OpenRules.com

DECISION CAMP 2023
Sep 18 - Sep 20, 2023

Motivation

- In 1997 Prof. Gene Freuder specified “the Holy Grail of Computer Science”:

> The user **defines** the problem, and the computer **solves** it!

- It points us to the Declarative Approach when
  - The user concentrates on Problem Definition
  - The computer does Problem Resolution

- How does it work in the Decision Modeling world today?

Decision Modeling - Procedural

- Rule-based movement started with the **Declarative approach** 40 years ago using RETE-based Rule Engines
- However, in the last 20 years Sequential Rule Engines have been used in the most practical rules-based decision-making applications
- Nowadays **Procedural approach** dominates Decision Modeling:
  - Decision models use rules to specify not only WHAT the decisioning rules are but also HOW to find a decision
  - Most modern DMN-like products provide their users with programming constructs and (explicitly or not) incentivize them to define decision-finding algorithms in rules

Decision Modeling - Declarative

- **What does constitute Declarative Decision Modeling**?
  - Concentration on “WHAT” and not on “HOW”
  - Decision Models mainly specify decision variables, relationships between them, and business objectives
  - Reliance on the *predefined* constraints and search methods to reach the decision model objective
  - A Declarative Decision Engine
    - Should not force a user to describe ALL possible situations in rules
    - It should be able to find a good or optimal decision automatically!

Sample Decision Model: Flight Rebooking

- This problem was proposed as a Decision Management Community Challenge in 2016

A flight was cancelled, and we need to re-book passengers to other flights considering their frequent flyer status, miles, and seat availability. Here is a sample data and flight assignment rules:

Flight	From	To	Dep	Arr	Capacity	Status
UA123	SFO	SNA	1/1/07 6:00 PM	1/1/07 7:00 PM	5	cancelled
UA456	SFO	SNA	1/1/07 7:00 PM	1/1/07 8:00 PM	2	scheduled
UA789	SFO	SNA	1/1/07 9:00 PM	1/1/07 11:00 PM	2	scheduled
UA1001	SFO	SNA	1/1/07 11:00 PM	1/2/07 5:00 AM	0	scheduled
UA1111	SFO	LAX	1/1/07 11:00 PM	1/2/07 5:00 AM	2	scheduled

Name	Status	Miles	Flight
Jenny	gold	500000	UA123
Harry	gold	100000	UA123
Igor	gold	50000	UA123
Dick	silver	100	UA123
Tom	bronze	10	UA123

Flight Rebooking:
Procedural Approach

- Most of the submitted solutions used different implementations of the following greedy algorithm:

Algorithm to build passenger-flight assignments:

1. First, sort all passengers using their GOLD, SILVER or BRONZE status. If two passengers have the same status use miles as a tiebreaker.
2. Repeat for every passenger from the sorted list:
   - Build a list of "suitable flights" for the selected passenger. A "suitable" flight should have the same departure and arrival airports as the cancelled flight and it also should still have an available seat
   - Sort the flights inside this list by an earlier departure time
   - Assign the flight on the top of the list to the current passenger
   - Decrement the flight's capacity

- Apparently, this is a Procedural Decision Model. It concentrates on HOW to find a decision
- This algorithm may find a decision, but it may not be the best one

Flight Rebooking: **Declarative Approach**

Given

F set of flights

P set of passengers from the canceled flight

For every passenger $p \in P$ and flight $f \in F$ Determine

$x_{pf} \in \{0,1\}$ = 1 if passenger *p* is assigned to flight $f \in F$

= 0 if otherwise

$delay_{pf}$ = number of hours between arrivals of the flight *f* and the passenger *p*'s canceled flight

= 100 if the passenger *p* is assigned to not scheduled flight *f*

$penalty_{pf}$ = $delay_{pf}$* $penaltyPerDelayedHour_p$

Subject to constraints

Each Passenger can be assigned to no more than 1 flight:

$x_{pf1} + x_{pf2} + \ldots + x_{pfn} <= 1$ for each passenger $p \in P$

Number of passengers assigned to the same flight cannot exceed the flight's capacity

$x_{p1f} + x_{p2f} + \ldots + x_{pnf} <= f_{capacity}$ for each flight $f \in F$

Minimize

$\sum_{p \in P} \sum_{f \in F} (penalty_{pf} * x_{pf})$ => MIN

- This model defines constraints for unknown decision variables x_{pf}, $delay_{pf}$, $penalty_{pf}$
- Objective is to minimize the total penalty, but it says nothing about "HOW" to do it

Decision Engine Implementations

- Decision Engines execute Decision Models
- Implementation techniques:
  1. Rule Engines:
     - Inferential (RETE)
     - Sequential (most DMN implementations)
     - Usually oriented to Business Users
  2. Use of LLMs to generate problem-specific decision engines
     - Natural language as an input
  3. Pure Constraint Solvers
  4. Integrated Rule Engine and Constraint Solver

Outline of my presentation

- Integrated use of Rule Engines and Constraint Solvers for declarative decision modeling
- Different Integration Approaches:
  1. Rule Engine implemented using a Constraint Solver
  2. Loosely coupled Decision Services:
     - *Business* Decision Service: Rules-based
     - *Technical* Decision Service: Constraint-based
  3. Using Rule-based and Constraint-based decision tables **inside the same Decision Model** (New)
- Sample Decision Models with Rule Solver

Rule Engines
(within Decision Management Environments)

- Efficiently execute Rules-based Decision Models for complex *business* problems
- Decision modeling is done using a user-friendly IDE that allows business(!) users to:
  - Create and maintain decision models using business rules in user-friendly formats such as standardized decision tables (**DMN**)
  - Define Rule Flows
  - Test and Debug Business Rules
  - Deploy Decision Models on-cloud or on-premise as Decision Services
- Rule Engine:
  - finds only one decision (not necessarily an optimal one)
  - requires everything to be defined in rules including both “What” and “How”
- **Usually Oriented to Subject Matter Experts**

Constraint Solvers

- Efficiently execute constraint-based Decision Models for complex *optimization* problems
- Constraint Solvers:
  - Implemented as:
    - Specialized Constraint Programming languages such as CPLEX OPL, AMPL, MiniZinc, or JSR331
    - API for C++, Java, or Python
  - Include predefined Global Constraints and Search Strategies
  - Capable to find Multiple and Optimal decisions
  - Frequently rely on a predefined search strategy not forcing a user to specify "How"
- **Usually Oriented to Software Developers**

Comparing Rule Engines and Constraint Solvers

Features	Rule Engine	Constraint Solver
Target Audience	Business Analysts (SMEs)	Software Developers
WHAT: Specifying Goals and Relationships	Decision Tables and other business-friendly DMN-like constructs	A programming language or a special CP modeling language (CPLEX OPL, AMPL, MiniZinc, or JSR331)
HOW: Search strategy to find a decision	- Required for commonly used sequential engines - Not required for inferential engines (rarely used nowadays)	- Usually not required - Heuristics may be configured to expedite the search
Decision Optimality	NO (in most practical cases)	YES (in many practical cases)

Using Constraint Solver as Rule Engine

- Key objectives:
  - Extend DMN to handle "unknown variables" like "known variables"
  - Solve optimization problems
  - Make Constraint Solvers more accessible to business users
- Two known implementations:

2011	Jacob Feldman published a paper "*Representing and Solving Rule-based Decision Models with Constraint Solvers*". It became the foundation of OpenRules® Rule Solver: • Rule Solver took a Business Decision Model implemented in accordance with the TDM standard (a predecessor of DMN) • Converted it to a Constraint Satisfaction Problem using the JSR331 standard representation • Used any off-the-shelf Constraint Solver included in JSR331 to validate and execute the decision model and find a feasible or optimal decision.
2020	KU Leuven scientists introduced an extension to the DMN standard called cDMN (Constraint Decision Model and Notation): • cDMN solves optimization-related Decision Management Community challenges using a DMN-like notation

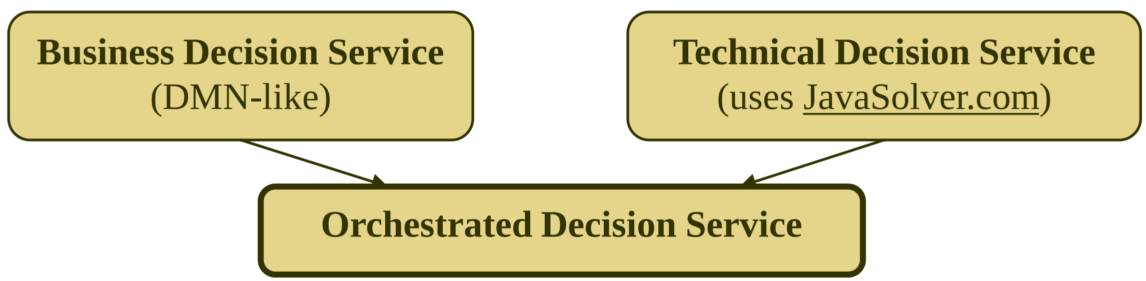
Integration Approach 1:
Constraint Solver as a Rule Engine

- Rule Engine implemented using a Constraint Solver
  - Input: DMN-like Decision Model
  - Output: Constraint Satisfaction Problem (CSP)
  - Execution mechanism: an off-the-shelf constraint (or linear) solver
- Advantages:
  - Consistency validation of decision models (inside and across all decision tables)
  - Ability to solve optimization problems
- Limitations:
  - Cannot handle popular decision modeling (DMN) constructs such as multi-hit decision tables, aggregation functions, loops and more
  - Does not use the entire power of a constraint solver
  - Makes intuitive decision tables harder to understand.

Integration Approach 2:

Loosely Coupled Rule Engine and Constraint Solver

- In the last 5 years, many decision management vendors and users switched to loosely coupled Decision Microservices deployed on-cloud
  - Orchestration of these RESTful services became quite simple and not dependent on their underlying implementations
  - So, in 2019 I published a paper "*Business Decision Modeling with Rule Engines and CP/LP Solvers*" that advocates splitting a decision model into three parts (decision services):



- This approach remains practical and powerful with 2 issues:
  - Involvement of technical experts
  - Passing of data between Business and Technical services

Integration Approach 3 (New)

Using Rule-based and Constraint-based Decision Tables Together

- DMN-like decision tables usually combine Condition and Conclusion columns:

Decision DefineMedication					
Condition		Condition		Conclusion	
Patient Age		Patient Allergies		Recommended Medication	
>=	18			Is	Amoxicillin
<	18			Is	Cefuroxime
		Include	Penicillin	Is	Levofloxacin

- The key idea: What if we expand regular DMN-like decision tables with new types of conditions and conclusions supported by a Constraint Solver?
  - Example for "Flight Rebooking": For each Passenger and each Flight the following table will create a new Booking variable that can take the value 0 or 1

Business Action

Solver Action (starts with prefix "Solver")

Decision AddNewBooking			
Action	SolverVar		
Booking	Var Name	Min	Max
{{Passenger Name}}-{{Flight Number}}	Booking	0	1

Integration Approach 3 (New)

Using Rule-based and Constraint-based Decision Tables Together

- New RuleSolver.com allows the author of decision models to mix and match traditional DMN-like constructs with Solver constructs within the same decision table, e.g.

Business Condition

Solver Action

DecisionTable AddPenaltyVariable					
Condition		SolverExpressionToList			
Flight Number		List Name	Var Name	Oper	Value
Is Not	CANCELED	All Penalty Variables	Booking	*	Penalty Per Delayed Hour * Delay Hours
Is	CANCELED				Penalty Per Delayed Hour * 100

- It means we may use special conditions and actions inside regular single-hit and multi-hit decision tables to:
  - Define constrained variables and mix them with regular variables
  - Define and post predefined linear and global constraints on these decision variables
  - Solve the problem by using predefined search methods to find feasible or optimal solutions

Decision Modeling with Rule Solver

- A user needs to define two main tables
  - **"Define"** that defines the problem
  - **"Solve"** that solves the problem

- Table "**Define**" requires major decision modeling efforts to define:
  - All known and yet unknown decision variables
  - Relationships between them (constraints)
  - Optimization Objective (optional)

- Table "**Solve**" usually is small and relies on predefined solving methods such as "SolverFindSolution"

Typical main tables:

Decision **Define**
ActionExecute
Steps
DefineVariables
DefineExpressions
DefineConstraints
PostConstraints1
PostConstraints2
PostConstraints3
PostConstraints4

Decision **Solve**
ActionExecute
Steps
SolverFindSolution

A very simple example: Map Coloring

This challenge deals with map coloring. You need to use no more than 4 colors (blue, red, green, or yellow) to color six European countries: Belgium, Denmark, France, Germany, Luxembourg, and the Netherlands in such a way that no neighboring countries use the same color.

Problem Specific

Decision **Define**
ActionExecute
Steps
DefineCountryVariables
PostNeighboringCountriesConstraints

Decision **Solve**
ActionExecute
Steps
SolverFindSolution

Predefined in Rule Solver

New Column Types

Decision DefineCountryVariables		
SolverVar		
Var Name	Min	Max
Belgium	1	4
Denmark	1	4
France	1	4
Germany	1	4
Netherlands	1	4
Luxembourg	1	4

Decision PostNeighboringCountriesConstraints		
SolverVarOperVar		
Country	oper	Country
France	!=	Belgium
France	!=	Luxembourg
France	!=	Germany
Luxembourg	!=	Germany
Luxembourg	!=	Belgium
Belgium	!=	Netherlands
Belgium	!=	Germany
Germany	!=	Denmark

Execution results: Belgium[1] Denmark[1] France[2] Germany[3] Netherlands[2] Luxembourg[4]

What if we have only 3 colors?

It means we should allow some neighboring countries to be colored with the same colors. Here are the relative costs for such **rule violations**:

France – Luxembourg: $257

Luxembourg – Germany: $904

Luxembourg – Belgium: $568

Old Rules:

Decision PostNeighboringCountriesConstraints		
SolverVarOperVar		
Country	oper	Country
France	!=	Belgium
France	!=	Luxembourg
France	!=	Germany
Luxembourg	!=	Germany
Luxembourg	!=	Belgium
Belgium	!=	Netherlands
Belgium	!=	Germany
Germany	!=	Denmark

New Hard and Soft Rules:

Decision PostHardConstraints		
SolverVarOperVar		
Country	oper	Country
France	!=	Belgium
France	!=	Germany
Belgium	!=	Netherlands
Belgium	!=	Germany
Germany	!=	Denmark

Solver Action

Decision CreateSoftConstraints			
SolverVarOperVarSoft			
Country	oper	Country	Violation Cost
Luxembourg	=	Belgium	568
France	=	Luxembourg	257
Luxembourg	=	Germany	904

Solver Action

What if we have only 3 colors?

We may add soft constraints to the list “Constraint Violations”:

Decision AddSoftConstraintsToList	
SolverVarToList	
Country	Country
Constraint Violations	Luxembourg = Belgium
Constraint Violations	France = Luxembourg
Constraint Violations	Luxembourg = Germany

Calculate “Total Constraint Violation”:

Decision DefineTotalConstraintViolation	
SolverSum	
Sum	Variables
Total Constraint Violation	Constraint Violations

Solver Action

And find a solution that Minimizes “Total Constraint Violation”:

Decision MinimizeTotalConstraintViolation	
SolverOptimize	
Optimization Type	Objectve
Minimize	Total Constraint Violation

Solver Action

Total Constraint Violation [257]

Belgium: red

Denmark: red

France: green

Germany: blue

Netherlands: green

Luxembourg: green

A more complex example: Where is Zebra?

- Problem Description:

  1. There are five houses.
  2. The Englishman lives in the red house.
  3. The Spaniard owns the dog.
  4. Coffee is drunk in the green house.
  5. The Ukrainian drinks tea.
  6. The green house is immediately to the right of the ivory house.
  7. The Old Gold smoker owns snails.
  8. Kools are smoked in the yellow house.
  9. Milk is drunk in the middle house.
  10. The Norwegian lives in the first house.
  11. The man who smokes Chesterfields lives in the house next to the man with the fox.
  12. Kools are smoked in the house next to the house where the horse is kept.
  13. The Lucky Strike smoker drinks orange juice.
  14. The Japanese smokes Parliaments.
  15. The Norwegian lives next to the blue house.

- Decision Model methods “Define” and “Solve”

Decision **Define**
ActionExecute
Steps
DefineVariables
DefineExpressions
DefineConstraints
PostConstraints1
PostConstraints2
PostConstraints3
PostConstraints4

Decision **Solve**
ActionExecute
Steps
SolverFindSolution

Where is Zebra?

- Define Constrained Decision Variables and Expressions

Decision DefineVariables				
#	SolverVarArray			
Rule	Array Name	Var Names	Min	Max
1	Colors	Green Ivory Blue Red Yellow	1	5
2	People	Norwegian Ukrainian Japanese Englishman Spaniard	1	5
3	Drinks	Juice Tea Milk Water Coffee	1	5
4	Pets	Snail Dog Fox Horse ZEBRA	1	5
5	Cigarettes	Chesterfield Parliament Lucky OldGolds Kools	1	5

Decision DefineExpressions			
SolverExpressionVarOperValue			
Expresion Name	Var Name	Oper	Value
House Right of Ivory	Ivory	+	1
House Right of Fox	Fox	+	1
House Left of Fox	Fox	-	1
House Right of Horse	Horse	+	1
House Left of Horse	Horse	-	1
House Right of Blue	Blue	+	1
House Left of Blue	Blue	-	1

Solver Action

Where is Zebra?

- Post Simple Constraints

Decision PostConstraints1		
SolverVarOperVar		
X	oper	Y
Englishman	=	Red
Spaniard	=	Dog
Coffee	=	Green
Ukrainian	=	Tea
OldGolds	=	Snail
Kools	=	Yellow
Lucky	=	Juice
Japanese	=	Parliament
Green	=	House Right of Ivory

Decision PostConstraints2		
SolverVarOperValue		
X	oper	Y
Milk	=	3
Norwegian	=	1

Decision PostConstraints3
SolverAllDiff
Array
Colors
People
Drinks
Pets
Cigarettes

Predefined Global Constraint “AllDiff”

Where is Zebra?

- Define and Post Relational Constraints

Decision DefineConstraints			
SolverConstraintVarOperVar			
Constraint Name	Var	oper	Value
Kools are smoked in the house **Right** to the house where the horse is kept	Kools	=	House Right of Horse
Kools are smoked in the house **Left** to the house where the horse is kept	Kools	=	House Left of Horse
The man who smokes Chesterfields lives in the house **Right** to the man with the fox	Chesterfield	=	House Right of Fox
The man who smokes Chesterfields lives in the house **Left** to the man with the fox	Chesterfield	=	House Left of Fox
The Norwegian lives **Right** to the blue house	Norwegian	=	House Right of Blue
The Norwegian lives **Left** to the blue house	Norwegian	=	House Left of Blue

Decision PostConstraints4	
SolverOr	
Constraint 1	Constraint 2
Kools are smoked in the house **Right** to the house where the horse is kept	Kools are smoked in the house **Left** to the house where the horse is kept
The man who smokes Chesterfields lives in the house **Right** to the man with the fox	The man who smokes Chesterfields lives in the house **Left** to the man with the fox
The Norwegian lives **Right** to the blue house	The Norwegian lives **Left** to the blue house

- Solution

Green[5] Ivory[4] Blue[2] Red[3] Yellow[1]

Norwegian[1] Ukrainian[2] Japanese[5] Englishman[3] Spaniard[4]

Juice[4] Tea[2] Milk[3] Water[1] Coffee[5]

Snail[3] Dog[4] Fox[1] Horse[2] **ZEBRA[5]**

Chesterfield[2] Parliament[5] Lucky[4] OldGolds[3] Kools[1]

Constraint-based Columns for Standard Decision Tables

- New constraint-based columns start with the prefix “**Solver**”
- Columns that **Define Constrained Variables**:

SolverVar		
Var Name	Min	Max

- Define a constrained variable with a domain Min-Max

SolverExpression			
Expresion Name	Var Name	Oper	Value

- Defines a constrained expression, e.g. “Var < 10”

SolverVarToList	
List Name	Var Name

- Adds a constrained variable to a list

SolverExpressionToList			
List Name	Var Name	Oper	Value

- Defines a constrained expression and adds it to a list

SolverSum	
Sum Name	List of Variables

- Defines a sum of variables

SolverScalarProduct		
Scalar Product Name	List of Variables	List of Coefficients

- Defines a scalar product of variables and coefficients

Constraint-based Columns for Standard Decision Tables

- Columns that **Post Constraints**:

SolverVarOperVar		
Var1 Name	Oper	Var2 Name

- Posts constraint "Var 1 <oper> Var 2", e.g. *"Total Calculated Discount <= 10% of Premium"*

SolveAllDiff
List of Variables

- Posts constraint which ensures that no two variables inside the provided list can have the same value

SolverSumOperValue		
List of Variables	oper	Value

- Posts constraint "Sum of List of Variables <oper> Value"

SolverScalarProductOperValue			
List of Variables	List of Coefficients	oper	Value

- Posts constraint "Variables X Coefficients <oper> Value"

SolverVarOperValueSoft			
Var	Oper	Value	Violation Cost

- Posts soft constraint: "Var <oper> Value with Violation Cost"

SolverIfThen					
Var1	Oper	Var2	Var1	Oper	Var2

- Posts constraint: **If** ("Var1 <oper> Var2") **Then** ("Var3 <oper> Var4")

SolverOr					
Var1	Oper	Var2	Var3	Oper	Var4

- Posts constraint: **Either** ("Var1 <oper> Var2") **Or** ("Var3 <oper> Var4")

Constraint-based Columns for Standard Decision Tables

- Predefined **Search Methods and Templates**:

SolverFindSolution

- This method finds a feasible solution

SolverSetObjective
Objective Name

- This template defines an Optimization Objective

SolverMinimize

- This method finds a solution that minimizes an already-defined objective

SolverOptimize	
Minimize or Maximize	Objective Variable

- This template finds a solution that Minimizes or Maximizes the Objective Variable

SolverSaveSolution

- This method saves a solution by assigning values of all instantiated variables to their business counterparts

SolverAssign	
Business Variable Name	**Solver Variable Name**

- This template assigns a found value of a “Solver Variable” to a “Business Variable”

How Rule Solver Is Implemented

- OpenRules provides an easy way to create custom columns for the standard decision tables
- Example:
  - Column “SolverOptimize”:

SolverOptimize	
Minimize or Maximize	Objective Variable

  - It is based on this template:

TemplateAction SolverOptimize	
solver(decision).optimize(type, objective);	
String type	String objective
Minimize or Maximize	Objective Variable

- Rule Solver utilizes an open-source “Java Constraint Programming API” (JSR-331)
- It can use any off-the-shelf Constraint Solver from JSR-331 without any changes in the decision model

Flight Rebooking Implementation (1)

- The complete decision model "Flight Rebooking" is described at http://RuleSolver.com
- This decision model is relatively complex for a live presentation, but here are a few implementation examples:

Decision Define	
ActionExecute	
Decision Tables	
CalculateBookingPenalties	for each passenger
DefineFlightSuitability	for each flight
DefineBookingVariables	for each passenger and each flight
PostAssignmentConstraints	
PostCapacityConstraints	

Decision Solve
ActionExecute
Decision Tables
DefineOptimizationObjective
MinimizeTotalPenalty
SetRebookings

Flight Rebooking Implementation (2)

- The most interesting part of this model:

$$penalty_{pf} = delay_{pf} * penaltyPerDelayedHour_{p}$$

Decision CalculateBookingPenalties [for each Passenger in Passengers]			
Condition	Condition	Conclusion	
Passenger Status	**Passenger Miles**	**Penalty Per Delayed Hour**	
		=	10
GOLD		+	15
SILVER		+	8
BRONZE		+	5
	500000+	+	4
	[250000..500000)	+	3
	[100000..250000)	+	2
	[25000..100000)	+	1

- If we decide to also consider a “Number of Traveling Children”, we will simply add another column to this business decision table (no changes in the solver part are required!)

Flight Rebooking: Defining Penalty Variables

- **For each Passenger and for each Flight**
  - Create a Booking constrained variable:

Decision AddNewBooking					
Action	Action		SolverVar		
Booking	Bookings		Var Name	Min	Max
{{Passenger Name}}-{{Flight Number}}	Add	Booking	Booking	0	1

  - Define Delay Hours:

DecisionTable DefineDelayHours
Action
Delay Hours
:= (int) Dates.hours(${Original Arrival Time}, ${Flight Arrival Time})

  - Create “All Penalty Variables”:

DecisionTable AddPenaltyVariable					
Condition		SolverExpressionToList			
Flight Number		List Name	Var Name	Oper	Value
Is Not	CANCELED	All Penalty Variables	Booking	*	Penalty Per Delayed Hour * Delay Hours
Is	CANCELED				Penalty Per Delayed Hour * 100

- **Then we will define ”Total Penalty”:**

Decision DefineOptimizationObjective	
SolverSum	
Sum Name	Variables
Total Penalty	All Penalty Variables

- **And minimize it using the predefined method “SolverOptimize”:**

DecisionTable MinimizeTotalPenalty	
SolverOptimize	
Optimization type	Objective
Minimize	Total Penalty

Future Improvements

- Current implementation with Solver Columns allows a user to concentrate on Problem Definition but it still uses too many low-level details

- Future improvement steps:
  - Offer more user-friendly constructs for Problem Definition
  - Move declarations of Solver variables and their relationships into the extended Business Glossary
  - Instead of using custom column templates, automatically generate Solver’s code
  - Potential integration with LLMs

Conclusion

- An advanced OpenRules Rule Solver integrates Rule Engine and Constraint Solver to support **Declarative Decision Modeling:**
  - Resulting decision models only specify Problem Definition (“What”)
  - Predefined Problem Resolution rules allow a decision model’s author not to worry about decision search (“How”)
- Side Effects of a new Rule Solver:
  - Instead of one possible decision, your decision model can find ***multiple*** and even ***optimal*** decisions
  - It makes traditional Constraint Solvers *business friendlier* using the expressive power of decision tables.

Thank you!

QnA

www.OpenRules.com