Passenger Rebooking - Decision Modeling Challenge

Solution by Edson Tirelli

## *Table of Contents*

*Table of Contents* 1

Introduction 1

Problem statement 2

Solution 2

Input Nodes 2

Prioritized Waiting List Decision 3

Passenger Priority Business Knowledge Model 4

Rebooked Passengers Decision 5

Reassign Next Passenger Business Knowledge Model 6

Has Capacity Business Knowledge Model 8

## Introduction

This is a solution to the DMN Community challenge from October/2016. This solution is strictly based on the **DMN specification**, compliance level 3. It can be executed using the **Drools open source engine** and it can be imported into **Trisotech**’s DMN Modeler for editing/authoring.

## Problem statement



## Solution

Following the DMN standard, the high-level solution is modelled in a DRD (Decision Requirements Diagram) that is presented below. Each node of the diagram is then explained in the following pages.

## Input Nodes

This problem statement defines two lists of input data:

* A list of flights, represented by the “Flight List” input node; Each flight contains all the flight attributes, as defined in the problem statement (i.e., flight number, from, to, etc)
* A list of passengers, represented by the “Passenger List” input node; Each passenger contains all the passenger attributes as defined in the problem statement (i.e., name, status, etc)

***Passenger Rebooking***

*Decision Requirements Diagram*

Rebooked Passengers

Prioritized Waiting List

passenger priority

reassign next passenger

has capacity

Passenger List

Flight List

## Prioritized Waiting List Decision

From the input data, the first decision finds the list of passengers from cancelled flights and sorts them in priority order, per the rules provided in the problem statement. This decision is modelled as a boxed context:

|  |  |
| --- | --- |
| **Prioritized Waiting List** | |
| Cancelled Flights | Flight List[ Status = **"cancelled"** ].Flight Number | |
| Waiting List | Passenger List[  **list contains**( Cancelled Flights,  Flight Number )  ] | |
| **sort**( Waiting List, passenger priority ) | | |

The first entry in the context uses a filter to find all the flights for which the status is “*cancelled*” and for each of those flights returns its “*Flight number*”. The result is a list of “*Flight Numbers*” that is assigned to the variable “*Cancelled Flights*”.

The second entry in the context also uses a filter to find all the passengers that were booked on the cancelled flights. This time, the filter uses a FEEL function named “*list contains()*” that returns true if the list passed as the first parameter (“*Cancelled Flights*” in this case) contains the element passed as the second parameter (“*Flight Number*” in this case). The result of this filter will be a list of all the passengers from the “*Passenger* *List*” that were booked into cancelled flights, and will assign that list to the variable “*Waiting* *List*”.

The final box in the boxed context is called result box and contains the expression that will be evaluated to produce the result of this decision. In this case, it is a call to another FEEL function called “*sort()*”. The “*sort()*” function will sort the “*Waiting List*” based on the “*passenger* *priority*” criteria (the second argument to the function) and will assign the resulting sorted list to the decision (i.e., the “*Prioritized* *Waiting* *List*” variable). The “*passenger* *priority*” criteria is modelled as a Business Knowledge Model and explained next.

## Passenger Priority Business Knowledge Model

The passenger priority for flight rebooking is defined by a composite criterion as per the problem statement. Basically, passengers with a higher status have priority (“gold” has priority over “silver” and “bronze”, etc) over passengers with a lower status. The problem statement is not clear on what happens if two passengers have the same status, but it is assumed that in this case, the passenger with a higher mileage will have priority.

This rule can be modelled in several ways, including FEEL expressions, decision tables or a combination of both.

In this case, a decision table is used, as it is typically considered to be more user friendly. It is modelled as a Business Knowledge Model node that receives two passengers as parameters and returns a boolean result of “true” if passenger 1 has priority over passenger 2, or false otherwise.

|  |  |  |
| --- | --- | --- |
| **passenger priority** | | |
| (Passenger1, Passenger2) | | |
| U | Passenger1.Status | Passenger2.Status | | Passenger1.Miles | Passenger1 has priority |
|  | *gold, silver, bronze* | *gold, silver, bronze* | |  | true, false |
| 1 | *gold* | *gold* | | > Passenger2.Miles | true |
| 2 | *silver, bronze* | | - | true |
| 3 | *silver* | *silver* | | > Passenger2.Miles | true |
| 4 | *bronze* | | - | true |
| 5 | *bronze* | *bronze* | | > Passenger2.Miles | true |

## Rebooked Passengers Decision

After creating a prioritized waiting list of passengers, the “Rebooked Passengers” decision reassign the passengers to new flights, depending on availability. It does that by invoking the Business Knowledge Model “reassign next passenger”.

The decision itself then returns a list of all the reassigned passengers as the solution for the problem.

|  |  |
| --- | --- |
| **Rebooked Passengers** | |
| reassign next passenger | |
| Waiting List | Prioritized Waiting List | |
| Reassigned Passengers List | [] | |
| Flights | Flight List | |

The parameters should be self-explanatory, except maybe the “Reassigned Passengers List”. This is a list of all the passengers already reassigned. On the first invocation, it starts empty, as one can see above. The empty square brackets (“[]”) is the FEEL representation for an empty list.

## Reassign Next Passenger Business Knowledge Model

The Business Knowledge Model “reassign next passenger” is a recursive function that will reassign all the passengers in the waiting list one by one. It is implemented as a boxed context for simplicity, and the explanation to each entry can be found after the diagram:

|  |  |
| --- | --- |
| **reassign next passenger** | |
| (Waiting List, Reassigned Passengers List, Flights) | |
| Next Passenger | Waiting List[1] | |
| Original Flight | Flights[ Flight Number = Next Passenger.Flight Number ][1] | |
| Best Alternate Flight | Flights[ From = Original Flight.From and   To = Original Flight.To and  Departure > Original Flight.Departure and  Status = **"scheduled"** and  **has capacity**( item, Reassigned Passengers List )  ][1] | |
| Reassigned Passenger | |  |  | | --- | --- | | Name | Next Passenger.Name | | Status | Next Passenger.Status | | Miles | Next Passenger.Miles | | Flight Number | Best Alternate Flight.Flight Number | | |
| Remaining Waiting List | **remove**( Waiting List, 1 ) | |
| Updated Reassigned Passenger List | **append**( Reassigned Passengers List, Reassigned Passenger ) | |
| **if**  **count**( Remaining Waiting List ) > 0  **then**  reassign next passenger( Remaining Waiting List,  Updated Reassigned Passengers List,            Flights ) **else**   Updated Reassigned Passengers List | | |

The BKM receives 3 parameters when it is invoked:

* **Waiting List:** the current list of passengers waiting for reassignment to a new flight
* **Reassigned Passengers List:** the list of passengers already reassigned
* **Flights:** the list of all available flights

The BKM evaluates the following context entries in order:

* **Next Passenger:** retrieves the next passenger to be assigned. It is the first passenger in the waiting list.
* **Original Flight:** retrieves the original flight the passenger was booked into by filtering the Flights list and finding the flight whose Flight Number is the same as the Flight Number of the passenger. A filter on a list returns a new list, so the use of [1] in the expression ensures that the result is a single element, not a list.
* **Best Alternate Flight:** finds the best alternate flight to assign the passenger to, based on the rules defined in the use case. I.e.:
  + the new flight must depart from the same location:
    - From = Original Flight.From
  + the new flight must arrive at the same location:
    - To = Original Flight.To
  + the new flight must depart after the original flight was scheduled to depart
    - Departure > Original Flight.Departure
  + the new flight must be scheduled (i.e., not cancelled)
    - Status = **"scheduled"**
  + the new flight must have a free seat for the new passenger. This condition is checked by invoking the “*has capacity()*” function (see its documentation in the next section for details), passing the flight as the first parameter and the already reassigned passengers list as the second parameter. It returns *true* if the flight still has a free seat for the new passenger, or *false* if it is already full.
    - **has capacity**( item, Reassigned Passengers List )
* **Reassigned Passenger:** creates a new record for the passenger with its new flight number. In case no flight was found matching the requirements in the previous entry, a null value is set on the *Flight* *Number* attribute.
* **Remaining Waiting List:** calculates the remaining waiting list by removing the passenger that was just reassigned from the original waiting list.
* **Updated Reassigned Passengers List:** updates the reassigned passengers list by appending the passenger that was just reassigned.
* **Result:** finally, the result box checks if the “*Remaining Waiting List*” is not empty, in which case it calls the “*reassign* *next passenger()*” function again to reassign the next passenger in the “*Remaining* *Waiting* *List*”. Otherwise, if the list is empty, it just returns “*Updated* *Reassigned* *Passengers* *List*” as the result of the invocation.

## Has Capacity Business Knowledge Model

The “*has capacity*” BKM is a simple function that checks if there are still free seats on a flight, considering all the passengers that were already reassigned to that flight.

It receives 2 parameters: the flight to check and a list of all rebooked passengers. It compares the flight remaining capacity with the number of passengers in the “rebooked list” already assigned to this flight. It returns true if there are still free seats or false otherwise.

|  |
| --- |
| **has capacity** |
| (flight, rebooked list) |
| flight.Capacity > **count**( rebooked list[ Flight Number = flight.Flight Number ] ) | |