OpenRules Solutions for Decision Model "Vacation Days Advanced"

In my closing remarks at DecisionCAMP-2018, I used the well-known decision model "Vacation Days" to explain the difference between two different approaches to decision modeling: Model-based vs. Method-based. All 20 decision models submitted as solutions for the Decision Management Community Jan-2016 Challenge were "method-based" meaning they expect a human modeler to describe exactly how to assign extra days avoiding possible conflicts. Read more about "Model-based vs. Method-based".

The recently published Nov-2018 Challenge deals with an advanced vacation days problem, for which a real "model-based" approach is more appropriate. Here is the problem definition:

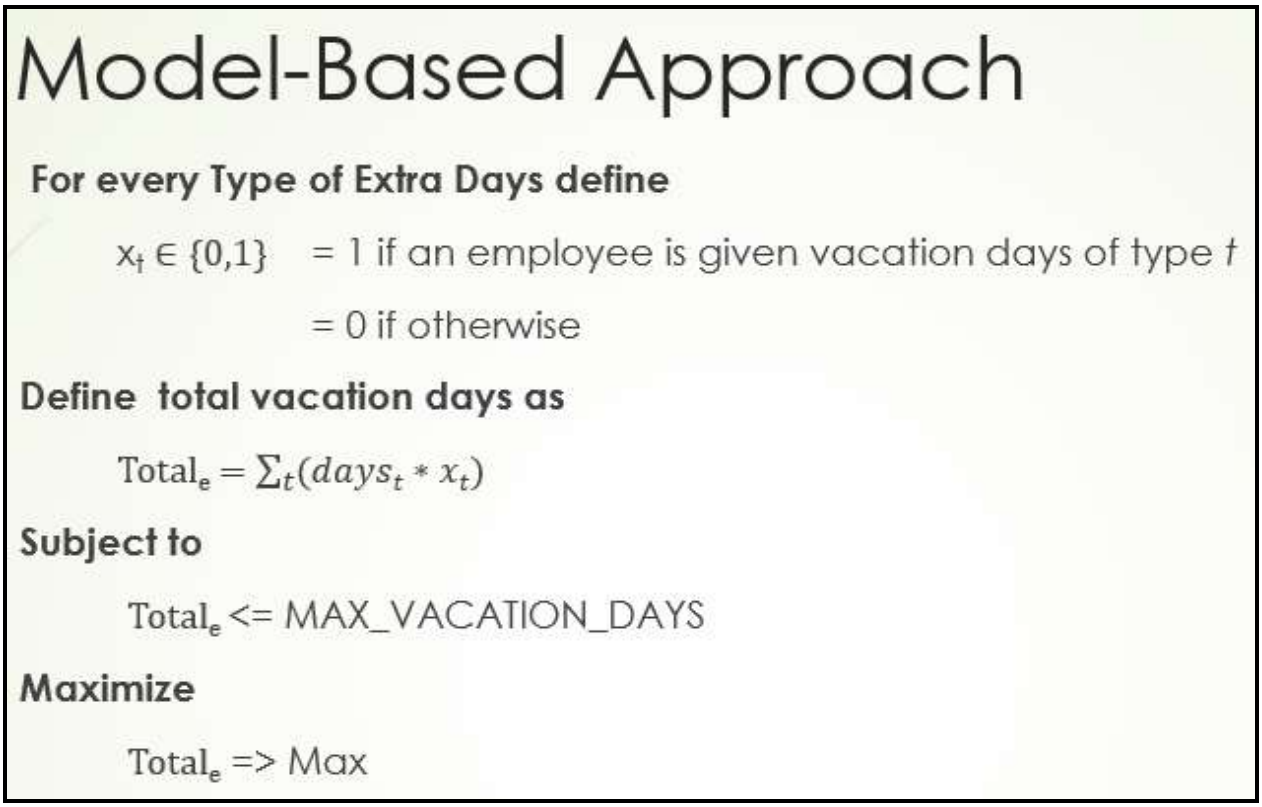
> *Every employee receives vacation days according to the following rules:*
> 1. *Every employee receives at least 22 vacation days.*
> 2. *Employees younger than 18 or at least 60 years, or employees with at least 30 years of service can receive extra 5 days.*
> 3. *Employees with at least 30 years of service and also employees of age 60 or more, can receive extra 3 days, on top of possible additional days already given*
> 4. *If an employee has at least 15 but less than 30 years of service, extra 2 days can be given. These 2 days can also be provided for employees of age 45 or more.*
> 5. *A college student is eligible to 1 extra vacation day.*
> 6. *If an employee is a veteran, 2 extra days can be given.*
> 7. *The total number of vacation days cannot exceed 29.*

I assume that the objective of this problem is still the same: give an employee as many vacation days as s/he is eligible to while satisfying all 7 rules. The rule 7 explicitly specifies the maximum of 29 vacation days that could be given to any employee. However, now not only Rule 2 and 4 can take us above the 29 days limit but many other combinations of extra days can do it as well. Of course, we still could use the "good old" method-based approach with three algorithmic steps:

1. *Calculate an employee's eligibility to all types of vacation days benefits*
2. *Apply all eligible criteria to calculate the total number of vacation days*
3. *If this number exceeds 29, assign 29 to the total number of vacation days*.

However, it would mean that we are allowed to subtract days from some types of eligible days and to give only a partial number of eligible days. What if it's not allowed? Consider a benefit management system with hundreds of possible benefit types (similar to vacation days) when some benefit types are mutually exclusive. The benefits could be associated with some values but they could not be given "partially" (everything or nothing). The method-based approach would not work in these cases, especially if you want to maximize the total value of all given benefits while providing some limitations (such as a percent of the total value).

Thus, we need to apply the model-based approach to this problem. Here is the proper model:



In this model, all eligibility rules still can be represented using business friendly decision tables such as 3 last tables in my old solution. However, we don't need any more a summary table "DefineVacationDays" with inter-rules relationships. Instead, we introduced unknown decision variables $\mathbf{x_t}$ to define if an employee will receive extra days of the type "t" or not. These variables are defined for every vacation days type including basic. Using these variables we can define the objective as a sum of all vacation days multiplied by these proper $\mathbf{x_t}$ variable. We also can limit this sum by some number MAX_VACATION_DAYS, e.g. 29. Then we can ask a decision engine to maximize this sum.

Below I describe two possible implementations of this model using OpenRules and JSR-331.

FIRST IMPLEMENTATION (Excel and Java)

Our future decision model will consist of two parts:

1) A pure business part that specifies employee eligibility to all types of vacation days
2) A more technical optimization part that deals with the Rule 7 and search of an optimal solution.

1. **Business Part**

The first part can be easily implemented using the following decision table:

DecisionTable DefineVacationTypeEligibility					
If	If	If	If	If	Then
Eligibility Type	**Age in Years**	**Years of Service**	**Student**	**Veteran**	**Eligibility Status**
Type1					TRUE
Type2	< 18				TRUE
Type2	>= 60				TRUE
Type2		>= 30			TRUE
Type3		>= 30			TRUE
Type3	>= 60				TRUE
Type4		[15..30)			TRUE
Type4	>= 45				TRUE
Type5			TRUE		TRUE
Type6				TRUE	TRUE
					FALSE

This is a very simple decision table (by default it is single-hit) that based on “Eligibility Type” and an employee’s “Age in Years”, “Years of Service”, and the checks “Student” and “Veteran” defines “Eligibility Status” as TRUE or FALSE. If all rules fail, the last unconditional rule will set “Eligibility Status” to FALSE. To complete this decision model, we only need to add the glossary:

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Age in Years	Employee	age
Years of Service		yearsOfService
Student		student
Veteran		veteran
Eligibilities		eligibilities
Total Vacation Days		totalVacationDays
Eligibility Type	Eligibility	type
Eligibility Status		status

I defined these table in the file “VacationDays.xls”. To test this model, I defined two Data tables in the file “Test.xls”. The first table

Data Employee employees				
name	age	yearsOf Service	student	veteran
Name	**Age in Years**	**Years of Service**	**Student**	**Veteran**
A	17	1	TRUE	FALSE
B	25	5	FALSE	FALSE
C	49	30	FALSE	TRUE
D	49	29	FALSE	FALSE
E	57	32	FALSE	TRUE
F	54	31	TRUE	FALSE
G	25	1	TRUE	TRUE
X	61	10	FALSE	FALSE

contains test-employees. The second table

Data Organization organizations			
id	maxVacationDays	vacationBenefitTypes	vacationBenefitDays
Id	**Max Vacation Days**	**Vacation Benefit Types**	**Vacation Benefit Days**
ABC	29	Type1	22
		Type2	5
		Type3	3
		Type4	2
		Type5	1
		Type6	2

describes an Organization with its 6 Vacation Benefit Types and associated Vacation Benefit Days. I decided to put here “Max Vacation Days” limit (29) as well.

Then I create two Java classes “Eligibility”, “Employee” and “Organization”. They are simple Java beans (a data structure) that correspond to the above glossary. Here are their attributes:

Eligibility:

```
String      type;
boolean     status;
int         assignedDays;
```

Employee:

```
String      name;
int         age;
int         yearsOfService;
boolean     student;
boolean     veteran;
int         totalVacationDays;
Eligibility[]eligibilities;
```

Organization:

```
String      id;
String[]    vacationBenefitTypes;
int[]       vacationBenefitDays;
int         maxVacationDays;
```

All getters and setters inside these classes were generated automatically using Eclipse IDE.

To execute our decision model against the tests I used the following Java launcher:

```
public static void main(String[] args) {
    // get data
    DecisionData data = new DecisionData("file:rules/Test.xls");
    Employee[] employees = (Employee[]) data.get("getEmployees");
    Organization organization = (Organization) data.get("getOrganization");
    for (int i = 0; i < employees.length; i++) {
        employees[i].setEligibilities(organization.getVacationBenefitTypes());
    }

    // create Decision
    DecisionModel model = new DecisionModel("file:rules/VacationDays.xls");
    Decision decision = model.createDecision("DefineVacationDaysEligibilities");

    // execute 'decision' for all employees
    for (int i = 0; i < employees.length; i++) {
        decision.log("\nTest " + (i+1));
        Employee employee = employees[i];
        decision.put("Employee", employee);
        decision.execute();
        employee.print();
    }
}
```

First, in the section “get data” we read employees and organization from the file Test.xls and initialize eligibilities for every employee. Then we create a decision model using the file “VacationDays.xls” and a decision “DefineVacationDaysEligibility”. Then we execute this decision for every employee. When we execute this model, it will print (e.g. for the first employee):

```
Test 1
Iterate over Eligibilities using rules DefineVacationTypeEligibility
  Assign: Eligibility Status = true
  Assign: Eligibility Status = true
  Assign: Eligibility Status = false
  Assign: Eligibility Status = false
  Assign: Eligibility Status = true
  Assign: Eligibility Status = false
Employee: name=A age=17 service=1 student=true veteran=false
Total Vacation Days=0
      type=Type1, status=true, assignedDays=0
      type=Type2, status=true, assignedDays=0
      type=Type3, status=false, assignedDays=0
      type=Type4, status=false, assignedDays=0
      type=Type5, status=true, assignedDays=0
      type=Type6, status=false, assignedDays=0
```

Here "`Total Vacation Days`" and "`assignedDays`" remain undefined (because we haven't implemented yet the second part).

2. Optimization Part

Now we will implement the second (optimization) part including Rule 7 and Search for an optimal solution. In the first implementation I decided to it directly in Java using JSR-331 (a constraint programming standard). I defined a Java class "Optimization" that looks as below:

```
public class Optimization extends OptimizationProblem {

    Organization organization;
    Employee employee;

    public Optimization(Organization organization, Employee employee) {
        super();
        this.organization = organization;
        this.employee = employee;
    }

    public void define() { // PROBLEM DEFINITION
        // Define assignment variables
        String[] types = organization.getVacationBenefitTypes();
        Var[] x = new Var[types.length];
        for (int i = 0; i < x.length; i++) {
            x[i] = csp.variable(types[i], 0, 1);
            if (!employee.isEligible(types[i]))
                csp.post(x[i],"=",0);
        }
        // Define optimization objective
        Var objective = csp.scalProd(organization.getVacationBenefitDays(),x);
        csp.add("TotalVacationDays", objective);
        setObjective(objective);

        // Limit
        csp.post(objective,"<=",organization.getMaxVacationDays());
    }

    public void saveSolution(Solution solution) {
        employee.setTotalVacationDays(solution.getValue("TotalVacationDays"));
        String[] types = organization.getVacationBenefitTypes();
        for (String type : types) {
            int assignedDays = organization.getVacationBenefitDays(type) * solution.getValue(type);
            employee.setAssignedDays(type, assignedDays);
        }
        solution.log();
    }
}
```

This class is a subclass of the standard OpenRules class "OptimizationProblem" that is used for optimization. We plan to create one object "optimization" for every pair "organization-employee". So, its constructor receives and memorizes the objects "organization" and "employee".

The method "define()" first defines an array decision variables "x" one for every type of vacation benefits. The snippet

```
x[i] = csp.variable(types[i], 0, 1);
if (!employee.isEligible(types[i]))
    csp.post(x[i],"=",0);
```

defines a constrained variable x[i] that can take values 0 or 1, and then forces it to be 0 if this employee is not eligible for this type of vacation benefits.

Then we define the optimization objective as a scalar product of an array x and vacation benefit days.

Then we limit the objective by `organization.getMaxVacationDays()`. This completes the definition of the constraint satisfaction problem.

We will rely on the default method "*solve()*" provided by the standard parent class OptimizationProblem. This method after find an optimal solution will call the method "*saveSolution(solution)*" which assigns found values back to the object "employee".

To execute both business and technical parts of our decision model, we need to add the optimization into our launcher that now will look as follows:

```
public static void main(String[] args) {

    // get data
    DecisionData data = new DecisionData("file:rules/Test.xls");
    Employee[] employees = (Employee[]) data.get("getEmployees");
    Organization organization = (Organization) data.get("getOrganization");
    for (int i = 0; i < employees.length; i++) {
        employees[i].setEligibilities(organization.getVacationBenefitTypes());
    }

    // create Decision
    DecisionModel model = new DecisionModel("file:rules/VacationDays.xls");
    Decision decision = model.createDecision("DefineVacationDaysEligibilities");

    // execute 'decision' for all employees
    for (int i = 0; i < employees.length; i++) {
        decision.log("\nTest " + (i+1));
        // Business problem
        Employee employee = employees[i];
        decision.put("Employee", employee);
        decision.execute();
        // Optimization problem
        OptimizationProblem opt = new Optimization(organization, employee);
        opt.define();
        opt.solve(Objective.MAXIMIZE);
        employee.print();
    }
}
```

Only the highlighted 4 lines were added. Note that we want to MAXIMIZE the optimization objective ("TotalVacationDays").

After executing this launcher, it will find optimal solutions for all 8 test-employees. For example, here are the execution results for the employee F:

```
Test 6
Iterate over Eligibilities using rules DefineVacationTypeEligibility
  Assign: Eligibility Status = true
  Assign: Eligibility Status = true
  Assign: Eligibility Status = true
  Assign: Eligibility Status = true
  Assign: Eligibility Status = true
  Assign: Eligibility Status = false
JSR-331 "Constraint Programming API" Release 1.2.2
JSR-331 Implementation based on Constrainer 5.4 (light)
=== SOLVE:
Found a solution with TotalVacationDays[0]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[1]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[2]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[3]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[4]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[5]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[6]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[7]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[8]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[9]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[10]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[11]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[22]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[23]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[24]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[25]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[26]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[27]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[28]. Wed Nov 14 22:11:21 EST 2018
Found a solution with TotalVacationDays[29]. Wed Nov 14 22:11:21 EST 2018
Solution #1:
         Type1[1] Type2[1] Type3[0] Type4[1] Type5[0] Type6[0] TotalVacationDays[29]
Employee: name=F age=54 service=31 student=true veteran=false
Total Vacation Days=29
        type=Type1, status=true, assignedDays=22
        type=Type2, status=true, assignedDays=5
        type=Type3, status=true, assignedDays=0
        type=Type4, status=true, assignedDays=2
        type=Type5, status=true, assignedDays=0
        type=Type6, status=false, assignedDays=0
```

You can see that the optimal solution was found after rejecting many other solutions with smaller objectives. Please note that while this employee is eligible to the vacation benefits of the Type3 (status=true) the assignedDays=0 because otherwise the Rule 7 will be violated.

This model for each type of eligible benefits assigns “all or nothing” – no partial days are allowed. Let’s modify the model to allow partial days for eligible benefit types. To do this, instead of using $\boldsymbol{days_t}$ as integers, we may define them as integer variables, which can take values from 0 to a number of days for the type t. The corresponding changes in the above class “Optimization” made to the methods “define” and “saveSolution”:

```
public void define() { // PROBLEM DEFINITION
    // Define assignment variables
    String[] types = organization.getVacationBenefitTypes();
    int[] days = organization.getVacationBenefitDays();
    Var[] dayVars = new Var[types.length];
    Var[] x = new Var[types.length];
    for (int i = 0; i < x.length; i++) {
        x[i] = csp.variable(types[i], 0, 1);
        if (!employee.isEligible(types[i]))
            csp.post(x[i],"=",0);
        Var dayVar = csp.variable(types[i]+"Days",0,days[i]);
        dayVars[i] = x[i].multiply(dayVar);
    }
    // Define optimization objective
    Var objective = csp.sum(dayVars);
    csp.add("TotalVacationDays", objective);
    setObjective(objective);
    // Limit
    csp.post(objective,"<=",organization.getMaxVacationDays());

}

public void saveSolution(Solution solution) {
    employee.setTotalVacationDays(solution.getValue("TotalVacationDays"));
    String[] types = organization.getVacationBenefitTypes();
    for (String type : types) {
        int assignedDays = solution.getValue(type+"Days");
        employee.setAssignedDays(type, assignedDays);
    }
    solution.log();
}
```

Now the objective is defined not as a scalar product but rather as a sum of the array of constrained variables dayVars:

```
Var objective = csp.sum(dayVars);
```

These `dayVars` were defined previously as products:

```
Var dayVar = csp.variable(types[i]+"Days",0,days[i]);
dayVars[i] = x[i].multiply(dayVar);
```

If we run our launcher again, the results for the employee F will look as follows:

```
Solution #1:
         Type1[1] Type1Days[18] Type2[1] Type2Days[5] Type3[1]
         Type3Days[3] Type4[1] Type4Days[2] Type5[1] Type5Days[1]
         Type6[0] Type6Days[0] TotalVacationDays[29]
Employee: name=F age=54 service=31 student=true veteran=false
Total Vacation Days=29
        type=Type1, status=true, assignedDays=18
        type=Type2, status=true, assignedDays=5
        type=Type3, status=true, assignedDays=3
        type=Type4, status=true, assignedDays=2
        type=Type5, status=true, assignedDays=1
        type=Type6, status=false, assignedDays=0
```

As you can see, the modified model assigned only 18 days from the 22 eligible days for the Type1, but gave all days for all other eligible types. It can be important in situations when the previous optimization model would give less than 29 vacation days.

This completes my first implementation. While this implementation demonstrates the power of the model-based approach to decision modeling, I am not satisfied (like probably many of my readers) with the fact that I used so much Java code.

SECOND IMPLEMENTATION (Excel without Java)

In my second implementation I tried to move Java code to Excel-based tables. I consider this as work in progress, and I am far from being satisfied with what I got so far. However, I've managed to move the optimization piece from Java to Excel, and it works now producing the same optimal results. So, I decided to share this preliminary representation of the same model hoping to get a constructive feedback from the readers. The generic (problem-independent) implementation logic is now hidden inside a special decision table template "DecisionTableCSPTemplate". So, here are my Excel-based tables (sorry, without comments as I hope they are self-explanatory).

Decision DetermineVacationDays
ActionExecute
Decision Tables
DefineEligibilityStatuses
CreateAssignVarArray
PostEligibilityConstraints
DefineTotalVacationDays
LimitTotalVacationDays
MaximizeTotalVacationDays

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Age in Years	Employee	age
Years of Service		yearsOfService
Student		student
Veteran		veteran
Eligibilities		eligibilities
Total Vacation Days		totalVacationDays
Eligibility Type	Eligibility	type
Eligibility Status		status
Vacation Benefit Types	Organization	vacationBenefitTypes
Vacation Benefit Days		vacationBenefitDays
Max Vacation Days		maxVacationDays

DecisionTable DefineEligibilityStatuses	
ActionIterate	
Array of Objects	**Rules**
Eligibilities	DefineVacationTypeEligibility

DecisionTable DefineVacationTypeEligibility					
If	If	If	If	If	Then
Eligibility Type	**Age in Years**	**Years of Service**	**Student**	**Veteran**	**Eligibility Status**
Type1					TRUE
Type2	< 18				TRUE
Type2	>= 60				TRUE
Type2		>= 30			TRUE
Type3		>= 30			TRUE
Type3	>= 60				TRUE
Type4		[15..30)			TRUE
Type4	>= 45				TRUE
Type5			TRUE		TRUE
Type6				TRUE	TRUE
					FALSE

DecisionTableCSP CreateAssignVarArray			
ActionCreateVarArray			
Var Array	**Using Names**	**Min**	**Max**
Variables	Vacation Benefit Types	0	1

DecisionTable PostEligibilityConstraints	
ActionIterate	
Array of Objects	**Rules**
Eligibilities	SetAssignVariables

DecisionTableCSP SetAssignVariables					
ConditionXoperY			ActionXoperY		
X <op> Y			Name	Oper	Value
Eligibility Status	=	FALSE	Eligibility Type	=	0

DecisionTableCSP DefineTotalVacationDays		
ActionScalarProduct		
Scalar Product Name	Coefficients	Variables
Total Vacation Days	Vacation Benefit Days	Variables

DecisionTableCSP LimitTotalVacationDays		
ActionXoperY		
X	<oper>	Y
Total Vacation Days	<=	Max Vacation Days

DecisionTableCSP	
ActionOptimize	
Optimization type	**Objective**
Maximize	Total Vacation Days