Business Decision Modeling with Rule Engines and CP/LP Solvers

Jacob Feldman

OpenRules, Inc., 53 Riviera Dr.,
Monroe, NJ 08831, USA
jacobfeldman@openrules.com

Abstract. Traditionally operational business decision models are represented and executed using business rules and decision management systems which utilize various rules engines. Such systems usually force human modelers to write rules for all possible combinations of decision variables to find one possible decision. In particular, this is true for decision models which follow the DMN standard. Alternatively, the same decision model can be represented using an optimization engine such as a constraint solver that does not need to know how to behave in all possible situations and is capable to automatically find multiple feasible decisions and even the optimal one. However, it is very difficult to represent all DMN constructs using only constraint programming facilities. Both these extreme approaches, rule engines and constraint solvers, have their limitations. This paper investigates the third way for business decision modeling when the entire decision model is split into several sub-models, some of which can be represented and executed using rule engines while others – using constraint or linear solvers. We demonstrate this approach using Decision Management Community Challenges and utilizing open source products "OpenRules" (for business decision models) and "Java Solver" (for optimization models) with various off-the-shelf constraint and linear solvers.

Keywords: Business Decision Modeling, Decision Optimization, DMN, Rule Engine, Constraint/Linear Solvers

1 Introduction

Nowadays business decision modeling is one of the major technological and methodological approaches that support decision-making processes across a wide range of business problems from loan origination and insurance underwriting to clinical guidelines and product recommendations. Business decision modeling is oriented to subject matter experts who work in concert with software developers to build and maintain operational decision models. In this paper we will consider 3 different approaches to business decision modeling:

1) Rules-based Decision Modeling approach that utilizes off-the-shelf business rules and decision management systems (BRDMS) based on different rules engines

2) Optimization-based Decision Modeling approach that utilizes an off-the-shelf constraint and/or linear solvers
3) Integrated Rules-based and Optimization-based approach that integrates both previous approaches.

1.1 Rules-based Approach

This approach is usually based on business rules management systems (BRMS), the majority of which transformed themselves into business rules and decision management systems (BRDMS) within the last 10 years. Nowadays business rules are considered mainly from the perspective of operational business decisions by representing the underlying decisioning logic. Correspondingly, rule engines started to be called “decision engines” while the majority of them still implement RETE-based and/or sequential execution mechanisms.

Different commercial and open-source BRDMSs are successfully used today for the creation and execution of real-world business decision models. They frequently utilize the decision modeling approach advocated by the Decision Modeling and Notation (DMN) standard [1] or go beyond it. However, these systems usually force human modelers to write business rules for all possible combinations of decision variables to specify how to find one and only one decision. They can't support the situations when a user is looking for several alternative decisions or optimal decisions that minimize/maximize certain business optimization objectives. In particular, the rules-based approach cannot deal with such practical situations when we need to find a decision that satisfies as many rules as possible or minimizes the total rules violation [2].

1.2 Optimization-based Approach

This is an alternative approach to business decision modeling that implements decision engines using constraint or linear solvers instead of traditional rule engines while decision models remain mainly the same still being expressed in business rules. This approach is described in this 2011 paper [1] that specifies how to automatically transfer a business decision model into a constraint satisfaction problem and to solve it using an off-the-shelf constraint solver. Constraint-based decision engines immediately brought several advantages:

- Automatic validation of decision models for consistency and completeness across all used decision tables
- Instead of forcing a human decision modeler to specify rules for all possible combinations of decision variables, it became possible for a subject matter expert to define only those rules which s/he considers important, and let a decision engine within a limited time to find multiple feasible decisions that satisfy the specified rules and let a user choose the best one

- If a decision model specifies an optimization objective, it became possible to automatically find the optimal decision that minimizes or maximizes the specified objective.

However, the constraint-based business decision models were limited to those that followed the methodological approach described in the book "The Decision Model" [3]. In particular, as it was pointed out in [4], this approach cannot handle more complex decisioning constructs described in the DMN standard and frequently used by real-world decision models. In particular, it doesn't support frequently used multi-hit decision tables with aggregation functions such as scorecards not mentioning complex boxed expressions specified using the DMN FEEL language [1]. Further research had shown that it's not clear how to implement these constructs using only constraint solvers.

1.3 Integrated Rules-based and Optimization-based Approach

Both previous approaches have their pros and cons. So, it's only natural to consider an approach that integrates rules-based and optimization-based approaches. This paper investigates such approach by describing how the entire business decision model can be split into several sub-models, some of which can be represented and executed using traditional rule engines while others – using constraint or linear solvers. Such a combination of rule engines and optimization solvers takes advantages of both worlds and avoids the limitations imposed when only one mechanism for problem definition and resolution is selected.

This paper shows how the integrated use of business rule engines and optimization solvers (CP/LP) can help to build much more powerful and flexible business decision models. Section 2 provides a generic definition of the operational decision model that can be used with any execution mechanism. Section 3 describes how to split a decision model into business and optimization sub-problems and how to use them in concert to produce feasible and optimal business decisions. Section 4 provides an example of a decision model implemented using the integrated approach.

2 Decision Model as a Combination of Business and Optimization Problems

In [4] we defined **Decision Model** by a set of **variables**, $X = \{ X_1, X_2,\ldots, X_n \}$, and a set of rules, $R = \{ R_1, R_2,\ldots, R_m \}$. Each variable X_i has a nonempty **domain** D_i of possible **values**. Each rule R_i defines relationships between different variables from a subset of X and specifies the allowable combinations of values for that subset.
A **decision** can be considered as an assignment of values to all mentioned variables, $\{ X_i=v_i, X_j=v_j, \ldots\}$, that satisfies all the rules.

The decision model can produce multiple decisions. Some decision models require a decision that minimizes/maximizes an **objective function**, which usually depends on some variables from X.

While this definition of the Decision Model takes its roots in the classical definition of a constraint satisfaction problem (CSP) given by Russel and Norvig [5], here we do not assume anything about rules representation or an algorithm that can resolve the problem. Different decision models can choose their own representations for:

- Decision variables - using a business glossary or constrained variables
- Rules – using DMN-like decision tables or linear and non-linear constraints
- Resolution algorithm – using a RETE-based rules engine, a sequential algorithm, or a constraint/linear solver.

More than that, a decision model can consist of several sub-models which utilize different representation and execution mechanisms.

3 How to Split Decision Model into Business and Optimization Parts

While decision models can utilize traditional rule engines or constraint/linear solvers, both these execution mechanisms have their own limitations. To overcome these limitations it's only natural to ask the question is: **why not to use a combination of these mechanisms whenever they are appropriate?** Real-world experience shows that decision models can be sliced, diced, and re-integrated in the way that the customers want. Let's assume that we split the decision modeling problem into different sub-problems that can have two types:

- **Business Problem**: uses business rules to specify relationships between different decision variables, creates a business decision model and executes it using a rule engine that (if necessary) will prepare input data for the optimization problem;
- **Optimization Problem**: defines optimization variables and constraints and uses off-the-shelf constraint or linear solver to find the optimal solution of the problem.

Of course, in reality, there could be several sub-problems, which can feed each other. Using this integrated approach, we may take advantages of both worlds and avoid the limitations imposed when a single representation and execution mechanism.

We will demonstrate this approach, we will use the following open-source tools:

- For business problems: OpenRules [6], a popular business rules and decision management system

- For optimization problems: Java Solver [7], a newly introduced product that provides probably the simples Java API to various off-the-shelf constraint and linear solvers (based on the JCP standard JSR331[8]).

4 Demonstration

We will use a sample decisioning problem specified as a Challenge "VacationDays" at the Decision Management Community website.

4.1 Problem Definition

First, we will consider a simple decision model recently published as Nov-2018 Challenge. 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 an 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, an 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 for 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.*

The objective of this problem is to give an employee as many vacation days as s/he is eligible to while satisfying all 7 rules. Rule 7 explicitly specifies the maximum of 29 vacation days that could be given to any employee.

4.2 Formal Decision Model

We can define our decision model as follows:

For every Type of Extra Days define

$x_t \in \{0,1\}$ = 1 if an employee is given vacation days of type *t*

= 0 if otherwise

Define total vacation days as

$$Total_e = \sum_t (days_t * x_t)$$

Subject to

$Total_e$ <= MAX_VACATION_DAYS

Maximize

$Total_e$ => Max

This model uses unknown decision variables **s** if an employee will receive extra days of the type "t" or not. These variables are defined for every vacation days type including basic 22 days. Using these variables we can define the optimization objective as a sum of all vacation days multiplied by the variables $\mathbf{x_t}$ over all types. We also can limit this sum by some number MAX_VACATION_DAYS=29. Then we can ask a decision engine to maximize this sum.

Thus, our decision model can be split into 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 Rule 7 and search of an optimal solution.

So, we will define two corresponding Java classes “BusinessProblem” and “OptimizationProblem”.

4.3 Test Data

However, before we jump to implementation, we should define test data for our future decision model. The following Excel table in the OpenRules format describes a Company “ABC” with its 6 Vacation Benefit Types and the corresponding Vacation Benefit Days. It also includes the “Max Vacation Days” limit of 29 days.

Variable Company company			
id	maxVacationDays	vacationBenefitTypes	vacationBenefitDays
Id	**Max Vacation Days**	**Vacation Benefit Types**	**Vacation Benefit Days**
ABC	29	Basic	22
		AgeOrService5	5
		AgeOrService3	3
		AgeOrService2	2
		Student	1
		Veteran	2

The following table describes different employees of this company that will be used for testing of our decision model:

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

We will keep these tables in the Excel file "rules/Test.xls"

4.4 Java Beans

To support this organization of data and our future decision model, we may create several Java beans: "Eligibility", "Employee" and "Company". Here they are:

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

Employee:
        String          name;
        Company         company;
        int             age;
        int             yearsOfService;
        boolean         student;
        boolean         veteran;
        int             totalVacationDays;
        Eligibility[]   eligibilities;

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

All getters and setters inside these Java classes can be generated automatically using Eclipse IDE.

4.5 Business Problem

Our class "BusinessProblem" should define employee eligibility to all types of the Company's vacation days. It can be done using the following business-friendly 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). It examines "Eligibility Type", "Age in Years", "Years of Service", "Student" and "Veteran" to define "Eligibility Status" as TRUE or FALSE. If all rules fail, the last unconditional rule will set "Eligibility Status" to FALSE.

We can put this table in the file "rules/VacationDays.xls". To complete this simple decision model, we only need to add the table 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

Now our class BusinessProblem can be implemented as follows:

```
public class BusinessProblem {

    Employee employee;

    public BusinessProblem(Employee employee) {
        this.employee = employee;
    }

    public boolean define() {
        DecisionModel model = new DecisionModel("file:rules/VacationDays.xls");
        Goal goal = model.createGoal("DefineVacationTypeEligibility");
        goal.use("Employee", employee);
        for (Eligibility eligibility : employee.getEligibilities()) {
            goal.use("Eligibility", eligibility);
            goal.execute();
        }
        return true;
    }
}
```

Thus, we will define an instance of Business Problem for each employee, and will use its method define() to "DefineVacationTypeEligibility".

4.6 Optimization Problem

Our class "OptimizationProblem" should implement the second (optimization) part of our decision model that deals with Rule 7 and the search for an optimal solution. We can implement this class as a subclass of JavaSolver [7]. A possible implementation of the OptimizationProblem is presented below.

The constructor of the class OptimizationProblem receives two objects "company" and "employee". The method *define()* creates an array x of Var variables that represent of the assignment variable $\mathbf{x}_t$. It also defines an optimization objective as a scalar product of the company's vacation benefit days and the array "**x**". This variable cannot exceed the company's Max Vacation Days.

We do not need to specify how this problem will be solved as we can rely on the standard JavaSolver's method *maximize()*. We only need to specify the method *saveSolution()* that will ve called when an optimal solution is found to save the results back to our business object "employee".

```java
public class OptimizationProblem extends JavaSolver {

    Company company;
    Employee employee;

    public OptimizationProblem(Company company, Employee employee) {
        this.company = company;
        this.employee = employee;
    }

    public void define() { // PROBLEM DEFINITION
        // Define assignment variables
        String[] types = company.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(company.getVacationBenefitDays(),x);
        csp.add("TotalVacationDays", objective);
        setObjective(objective);
        // Limit
        csp.post(objective,"<=",company.getMaxVacationDays());
    }

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

4.7 Executing the Integrated Decision Model

To execute our integrated decision model against the above test cases we can use the following Java launcher:

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

    DecisionData data = new DecisionData("file:rules/Test.xls");
    Company company = (Company) data.getVariable("company");
    Employee[] employees = (Employee[]) data.get("employees");
    company.setEmployees(employees);

    for (int i = 0; i < employees.length; i++) {
        Employee employee = employees[i];
        System.out.println("\n=== Assign Vacation Days to Employee " + employee.getName());

        BusinessProblem businessProblem = new BusinessProblem(employee);
        businessProblem.define();

        OptimizationProblem optimizationProblem = new OptimizationProblem(company, employee);
        optimizationProblem.define();
        optimizationProblem.maximize();

        employee.print();
    }
}
```

First, it will get “company” and “employees” from the Excel file “rules/Test.xls”. Then for every employee, it will create an instance of BusinessProblem that will define the employee’s eligibility to different types of vacation days. Then it will create an instance of OptimizationProblem that will find an optimal solution.

Here are the execution results for the first test:

```
*** Execution Profile ***
Number of Choice Points: 13
Number of Failures: 2
Occupied memory: 35731248
Execution time: 3 msec
Employee: name=X age=61 service=10 student=false veteran=false
        type=Basic, status=true, assignedDays=22
        type=AgeOrService5, status=true, assignedDays=5
        type=AgeOrService3, status=true, assignedDays=0
        type=AgeOrService2, status=true, assignedDays=2
        type=Student, status=false, assignedDays=0
        type=Veteran, status=false, assignedDays=0
Total Vacation Days=29
```

More complex examples of decision models that utilize the integrated use of rule engines and CP/LP solvers are described at [9] and [10].

References

1. Decision Model and Notation™ (DMN™), Object Management Group (OMG), http://www.omg.org/spec/DMN/Current
2. J.Feldman: Representing and Solving Rule-Based Decision Models with Constraint Solvers, in Proc. of RuleML 2011 - America, LNCS 7018, pp. 208-221, 2011, Springer-Verlag Berlin Heidelberg 2011, http://www.springerlink.com/content/e66077u456548231
3. B. von Halle, L.Goldberg: The Decision Model: A Business Logic Framework Linking Business and Technology. Auerbach Publications/Taylor & Francis Group, LLC (2009)
4. J. Feldman: What-If Analyzer for DMN-based Decision Models, Proceedings of the RuleML 2016 Challenge, http://ceur-ws.org/Vol-1620/paper2.pdf
5. S.Russell, P.Norvig: Artificial intelligence: a modern approach, Third Edition, http://aima.cs.berkeley.edu/2nd-ed/newchap05.pdf
6. OpenRules, Open Source Business Rules and Decision Management System, http://openrules.com
7. Java Solver, http://javasolver.htm
8. Java Request Specification (JSR) 331: Constraint Programming API. Java Community Process, http://www.jcp.org/en/jsr/detail?id=331
9. Model-based Optimal Solutions for Flight Rebooking Challenge, OpenRules Blog Dec-2018, https://openrules.wordpress.com/2018/12/26/model-based-optimal-solutions-for-flight-rebooking-challenge/
10. Building Decision Models for DMCommunity.org Challenge “Balanced Assignment”, OpenRules Blog Apr-2019, https://openrules.wordpress.com/2019/04/11/building-decision-models-for-dmcommunity-org-challenge-balanced-assignment/