Using Decision Tables to Model and Solve Scheduling and Resource Allocation Problems

Presenter: Dr. Jacob Feldman

OpenRules Inc., CTO

bbc BUILDING BUSINESS CAPABILITY

Using Decision Modeling

"Every situation, no matter how complex it initially looks, is exceedingly simple"
Eli Goldratt

- Decision Modeling is a methodological (technology independent) approach that allows subject matter experts to build executable decision models
- We will demonstrate how a business analyst without programming background may build Scheduling Decision Models

Scheduling and Resource Allocation Problems

- Schedule Activities (jobs)and Assign Resources to them
- Traditionally considered as very complex business problems
- They usually out of reach for the most rule engines
- Frequently require an OR (Operation Research) guru with deep knowledge of C++/Java

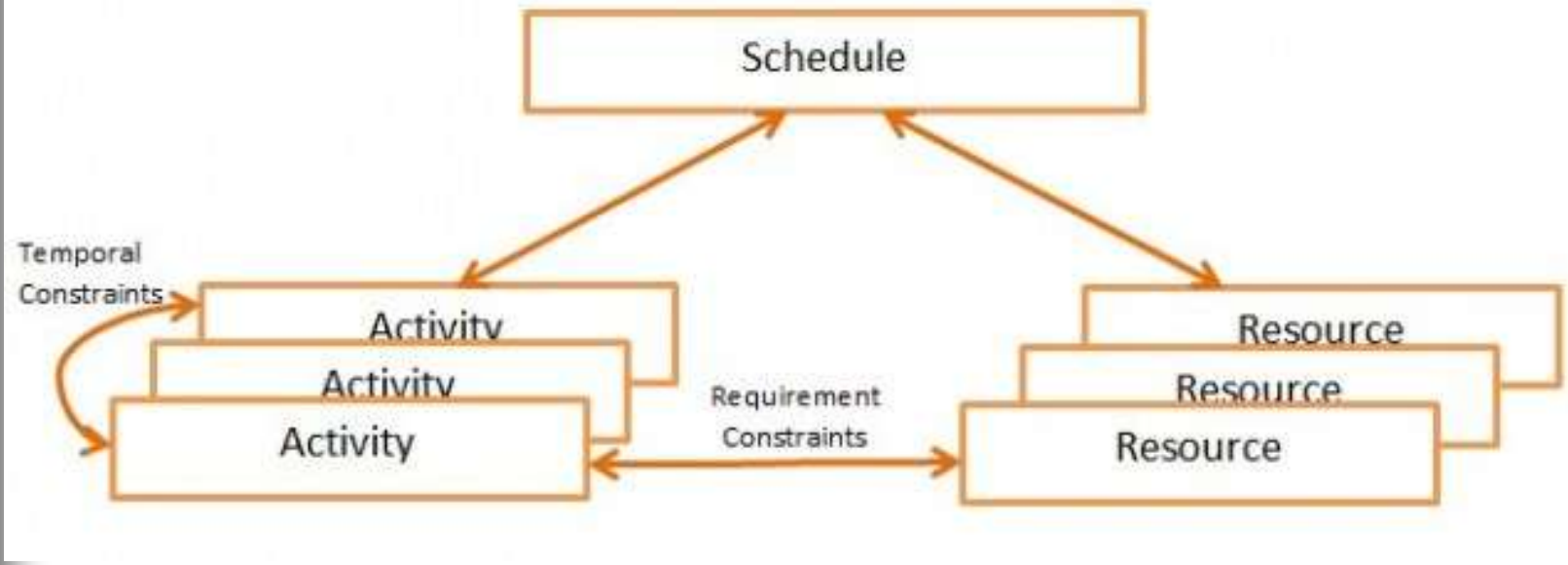
Real-world example: Workforce/Workload Management

- Field Service Scheduling for a major Gas and Electric Utility
- More than 1 million customers
- More than 5000 employees
- Large Service territory
- Hundreds small and large jobs per day
- Job requires a mix of people skills, vehicles and equipment

Multi-Objective Optimization

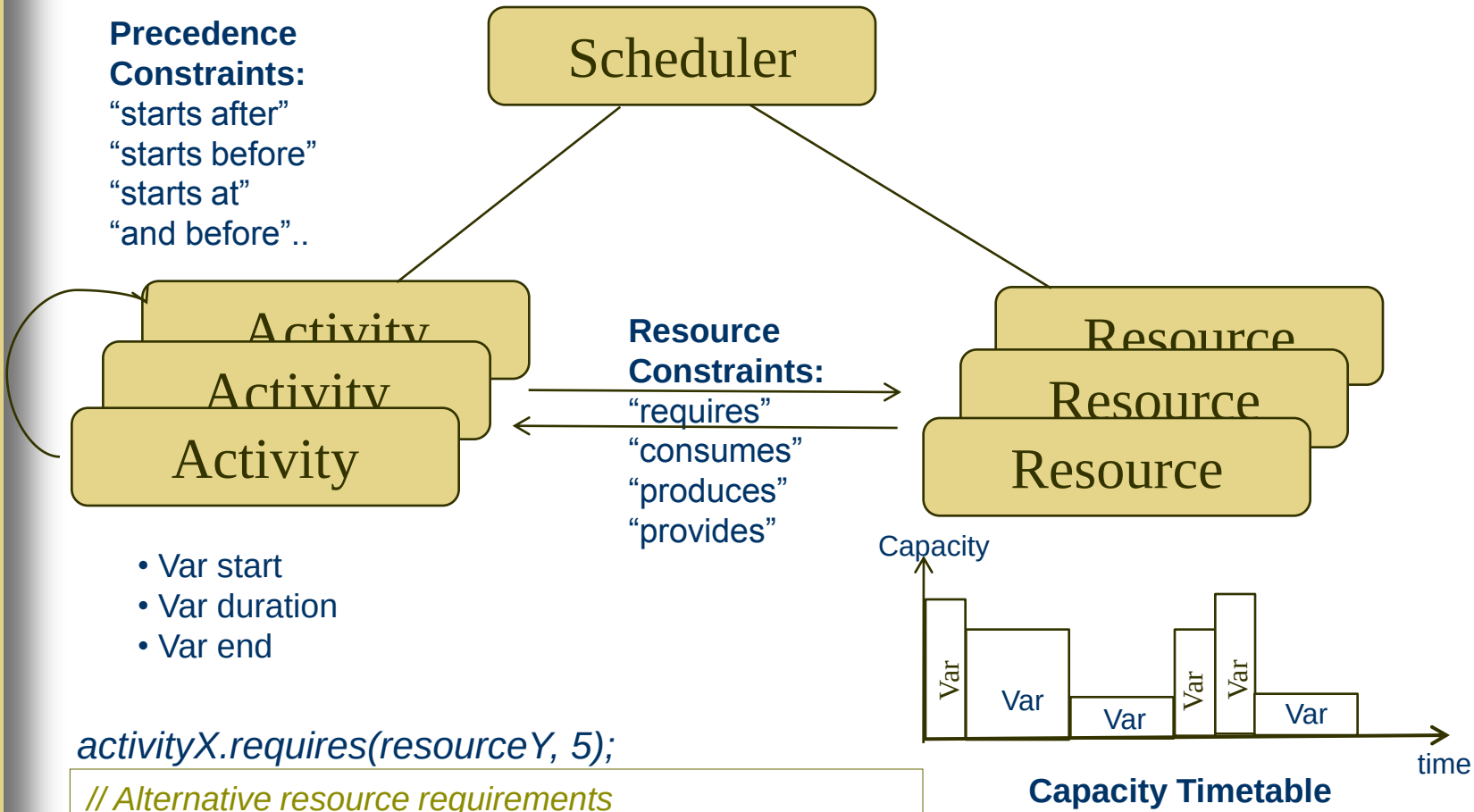
- Multi-objective Work Planning and Scheduling:
  - Travel time minimization
  - Resource load levelization
  - Skill utilization (use the least costly skills/equipment)
  - Schedule jobs ASAP
  - Honor user-defined preferences

Generic Model



- Activities (jobs) with yet unknown start times and known durations (not always)
- Resources with limited capacities varying over time
- Constraints
  - Between activities: Job2 starts after the end of Job1
  - Between activities and resources:
    Job1 requires a welder for 1 hour, Job 2 consumes $1,500

JSR331 Scheduler



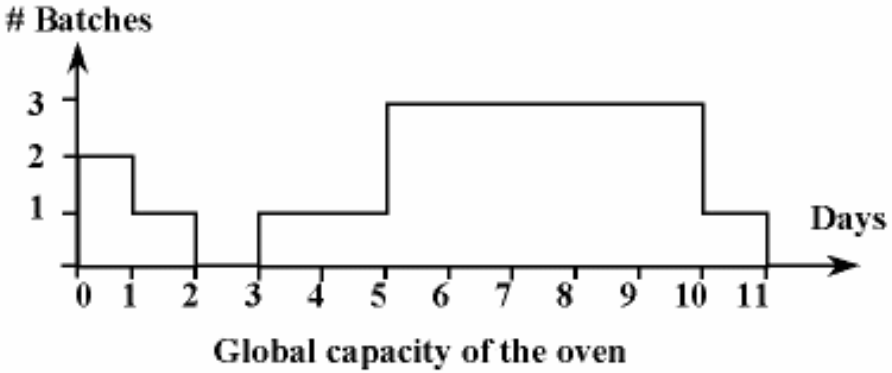
activityX.requires(resourceY, 5);

```
// Alternative resource requirements
activity1.requires(resource2, varReq2);
activity1.requires(resource3, varReq3);
varReq2.ne(varReq3);
```

Sample Problem

Oven - job scheduling with one resource

There is an oven in which we can fire batches of bricks. There are five orders to fire X batches during Y days. Schedule all orders to be done in no more than 11 days taking into consideration the following oven availability:



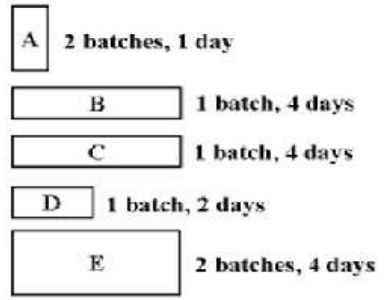
Global capacity of the oven

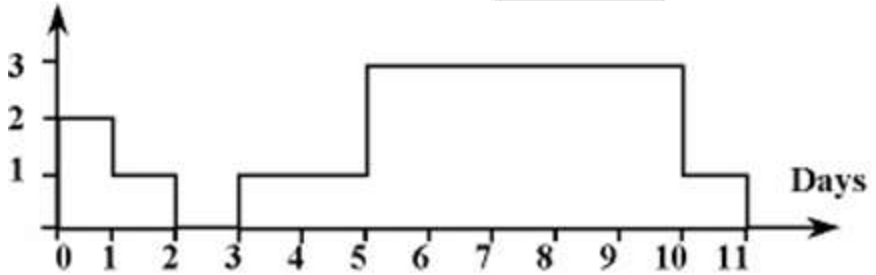
A	2 batches, 1 day
B	1 batch, 4 days
C	1 batch, 4 days
D	1 batch, 2 days
E	2 batches, 4 days

5 Activities

Sample Problem Implemetation
(Java with JSR-331 Scheduler)

```
Schedule schedule = ScheduleFactory.newSchedule("oven"0, 11);
Activity A = schedule.addActivity(1, "A");
Activity B = schedule.addActivity(4, "B");
Activity C = schedule.addActivity(4, "C");
Activity D = schedule.addActivity(2, "D");
Activity E = schedule.addActivity(4, "E");
Resource oven = schedule.addResource(3, "oven");
oven.setCapacityMax(0, 2);
oven.setCapacityMax(1, 1);
oven.setCapacityMax(2, 0);
oven.setCapacityMax(3, 1);
oven.setCapacityMax(4, 1);
oven.setCapacityMax(10, 1);
// Resource Constraints
A.requires(oven, 2).post();
B.requires(oven, 1).post();
C.requires(oven, 1).post();
D.requires(oven, 1).post();
E.requires(oven, 2).post();
// Find Solution
schedule.scheduleActivities();
schedule.displayActivities();
```





SOLUTION:
A[5 -- 1 --> 6) requires oven[2]
B[3 -- 4 --> 7) requires oven[1]
C[7 -- 4 --> 11) requires oven[1]
D[0 -- 2 --> 2) requires oven[1]
E[6 -- 4 --> 10) requires oven[2]

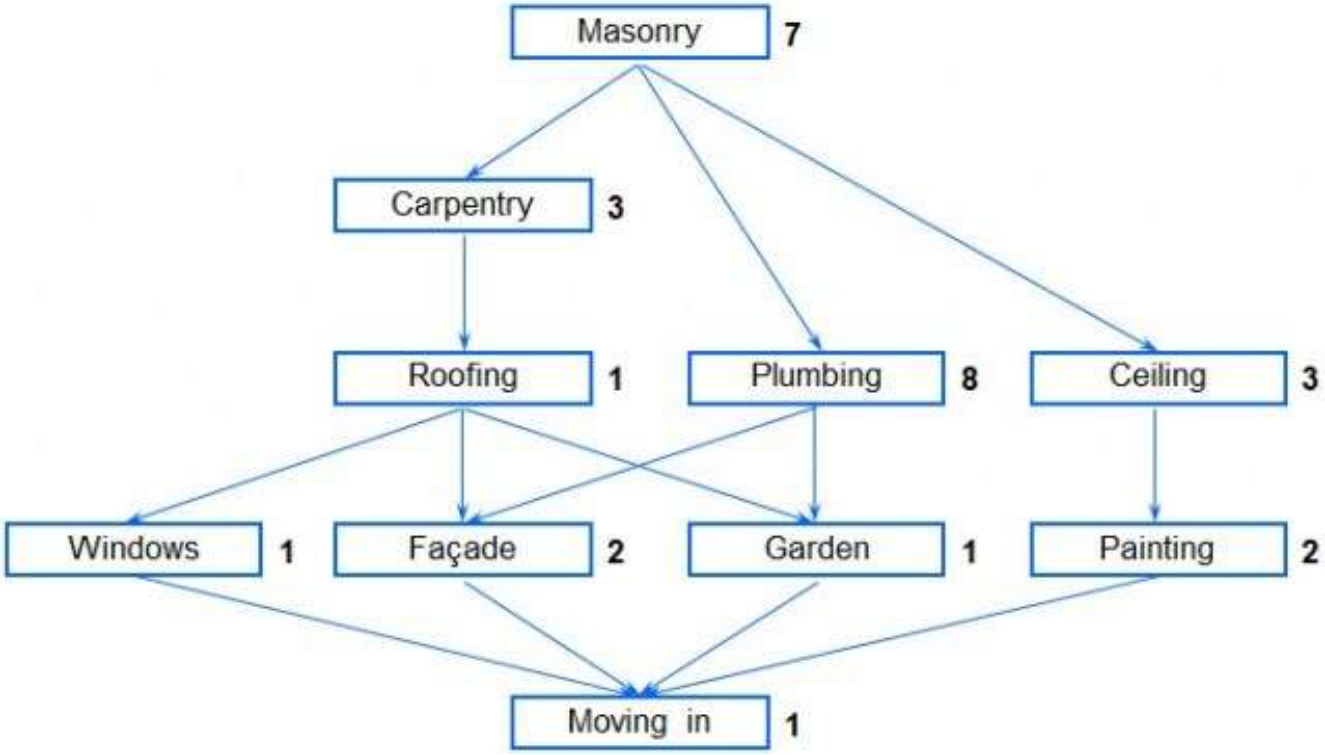
Open Source Software

- **JSR 331**: "Constraint Programming API" an official Java Community Process standard www.jcp.org
  - Comes with several open source implementations
  - Free Download: **http://openrules.com/jsr331**

- **Rule Solver**: a component of OpenRules, an open source Business Decision Management System www.openrules.com

Example "House Construction"

House construction requires the following activities with fixed durations and precedence constraints:



Objective: Move in ASAP!

Decision

Top-level Decision “Schedule Activities” with 3 sub-decisions:

Decision ScheduleActivities	
Decisions	**Execute Rules**
Define Schedule	:= DefineSchedule()
Define Activities	:= DefineActivities()
Define Precedence Constraints	:= DefinePrecedenceConstraints()

Sub-Decisions 1 and 2

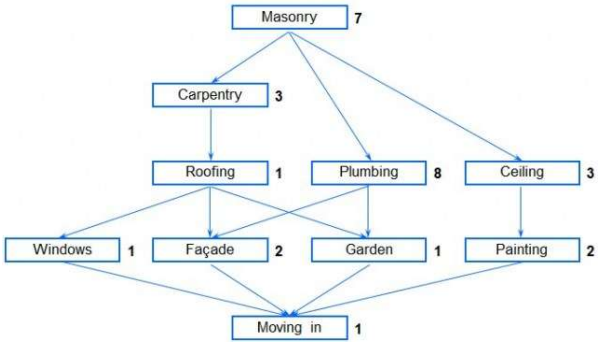
Define Schedule with a makespan 30 days:

DecisionTable DefineSchedule	
ActionSchedule	
Origin	**Horizon**
0	30

Define Activities:

DecisionTable DefineActivities	
ActionAddActivity	
Name	**Duration**
masonry	7
carpentry	3
roofing	1
plumbing	8
ceiling	3
windows	1
façade	2
garden	1
painting	2
movingIn	1

Sub-Decision 3



Define Precedence Constraints:

DecisionTable DefinePrecedenceConstraints		
ActionActOperAct		
Activity	**Operator**	**Activity/Day**
carpentry	>	masonry
roofing	>	carpentry
plumbing	>	masonry
ceiling	>	masonry
windows	>	roofing
façade	>	plumbing
façade	>	roofing
garden	>	roofing
garden	>	plumbing
movingIn	>	windows
movingIn	>	façade
movingIn	>	garden
movingIn	>	painting

The decision model is ready to be executed!

Execute Decision Model

Run Rule Solver:

```
masonry   [0 -- 7 --> 7)
carpentry [7 -- 3 --> 10)
roofing    [10 -- 1 --> 11)
plumbing [7 -- 8 --> 15)
ceiling     [7 -- 3 --> 10)
windows   [11 -- 1 --> 12)
façade      [15 -- 2 --> 17)
garden      [15 -- 1 --> 16)
painting    [0 -- 2 --> 2)
movingIn   [17 -- 1 --> 18)
```

Note: We actually defined “WHAT” and relied on the default “HOW”!

House Construction with a Worker

- Let's assume that all of the activities require a Worker in order to be processed
- Now we cannot schedule two activities at the same time because the Worker can perform only one task at a time!

Decision

We need to add two more sub-decisions (4) and (5) to our previous decision:

Decision ScheduleActivitiesWithWorker	
Decisions	**Execute Rules**
Define Schedule	:= DefineSchedule()
Define Activities	:= DefineActivities()
Define Precedence Constraints	:= DefinePrecedenceConstraints()
Define Worker	:= DefineWorker()
Define Resource Requirement Constraints	:= ResourceRequirementConstraints()

(4)

(5)

Sub-Decision 4

Define a worker as a recoverable resource with maximal capacity equal to 1 day:

DecisionTable DefineWorker		
ActionAddResource		
Name	**Type**	**Max Capacity**
Worker	Recoverable	1

Sub-Decision 5

Define resource constraints:
Each activity requires the resource "Worker":

DecisionTable ResourceRequirementConstraints		
ActionActReqResource		
Activity	**Required Resource**	**Required Capacity**
masonry	Worker	1
carpentry	Worker	1
roofing	Worker	1
plumbing	Worker	1
ceiling	Worker	1
windows	Worker	1
façade	Worker	1
garden	Worker	1
painting	Worker	1
movingIn	Worker	1

Execute Decision Model

Run Rule Solver:

masonry[0 -- 7 --> 7) requires Worker[1]
carpentry[7 -- 3 --> 10) requires Worker[1]
roofing[10 -- 1 --> 11) requires Worker[1]
plumbing[11 -- 8 --> 19) requires Worker[1]
ceiling[19 -- 3 --> 22) requires Worker[1]
windows[22 -- 1 --> 23) requires Worker[1]
façade[23 -- 2 --> 25) requires Worker[1]
garden[25 -- 1 --> 26) requires Worker[1]
painting[26 -- 2 --> 28) requires Worker[1]
movingIn[**28** -- 1 --> 29) requires Worker[1]

House Construction with a Worker and Limited Budget

- Now along with worker constraints, we have to consider budget constraints. Each activity requires the payment of $1,000 per day
- A bank agreed to finance the house construction for the total amount of $30,000. However, the sum is available in two installations:
  - $15,000 is available at the start of the project
  - $15,000 is available 15 days afterwards

Decision

We need to add two more sub-decisions (4) and (5) to our previous decision :

Decision ScheduleActivitiesWithWorkerBudget	
Decisions	**Execute Rules**
Define Schedule	:= DefineSchedule()
Define Activities	:= DefineActivities()
Define Precedence Constraints	:= DefinePrecedenceConstraints()
Define Worker & Budget	:= DefineResources()
Define Budget Limitations	:= SetBudgetCapacities()
Define Resource Requirement Constraints	:= ResourceRequirementConstraints()

(4)

(5)

(6)

Sub-Decision 4

Define two resources:

- a worker as a recoverable resource with maximal capacity equal to 1 day
- A budget as a consumable with maximal capacity $30,000

DecisionTable DefineResources		
ActionAddResource		
Name	**Type**	**Max Capacity**
Worker	Recoverable	1
Budget	Consumable	30000

Sub-Decision 5

We already specified maximal capacity of the resource budget. So, it is enough to specify the limit $15,000 for the first 15 days:

DecisionTable SetBudgetCapacities			
ActionResourceCap			
Resource	From	To	Capacity
Budget	0	15	15000

Sub-Decision 6

The extended table "ResourceRequirement Constraints":

DecisionTable ResourceRequirementConstraints		
ActionActReqResource		
Activity	**Required Resource**	**Required Capacity**
masonry	Worker	1
carpentry	Worker	1
roofing	Worker	1
plumbing	Worker	1
ceiling	Worker	1
windows	Worker	1
façade	Worker	1
garden	Worker	1
painting	Worker	1
movingIn	Worker	1
masonry	Budget	1000
carpentry	Budget	1000
roofing	Budget	1000
plumbing	Budget	1000
ceiling	Budget	1000
windows	Budget	1000
façade	Budget	1000
garden	Budget	1000
painting	Budget	1000
movingIn	Budget	1000

Execute Decision Model

Run Rule Solver:

masonry[0 -- 7 --> 7) requires Worker[1] requires Budget[1000]
carpentry[7 -- 3 --> 10) requires Worker[1] requires Budget[1000]
roofing[10 -- 1 --> 11) requires Worker[1] requires Budget[1000]
plumbing[11 -- 8 --> 19) requires Worker[1] requires Budget[1000]
ceiling[19 -- 3 --> 22) requires Worker[1] requires Budget[1000]
windows[22 -- 1 --> 23) requires Worker[1] requires Budget[1000]
façade[23 -- 2 --> 25) requires Worker[1] requires Budget[1000]
garden[25 -- 1 --> 26) requires Worker[1] requires Budget[1000]
painting[26 -- 2 --> 28) requires Worker[1] requires Budget[1000]
movingIn[**28** -- 1 --> 29) requires Worker[1] requires Budget[1000]

House Construction with Alternative Resources

- Let's assume that we have three workers Joe, Jim, and Jack with different skills
- Each job requires only one of these workers depending on their skills:

masonry	requires Joe or Jack
carpentry	requires Joe or Jim
plumbing	requires Jack
ceiling	requires Joe or Jim
roofing	requires Joe or Jim
painting	requires Jack or Jim
windows	requires Joe or Jim
façade	requires Joe or Jack
garden	requires Joe or Jack or Jim
movingIn	requires Joe or Jim.

Workers are alternative resources!

Decision

We need to add two more sub-decisions (4) and (5) to our previous decision :

Decision ScheduleActivitiesWithAlternativeResources		
Decisions	**Execute Rules**	
Define Schedule	:= DefineSchedule()	
Define Activities	:= DefineActivities()	
Define Precedence Constraints	:= DefinePrecedenceConstraints()	
Define Workers	:= DefineWorkers()	(4)
Define Resource Requirement Constraints	:= ResourceRequirementConstraints()	(5)

Sub-Decision 4

The sub-decision “Define Workers” adds 3 alternative resources that should be specified as special “disjunctive” resources:

DecisionTable DefineWorkers		
ActionAddResource		
Name	**Type**	**Max Capacity**
Joe	disjunctive	1
Jack	disjunctive	1
Jim	disjunctive	1

Sub-Decision 5

This decision table is similar to previous "ResourceRequirementConstraint" tables but lists different alternative resources separated by the OR-sign "|":

DecisionTable ResourceRequirementConstraints		
ActionActReqResource		
Activity	**Required Resource**	**Required Capacity**
masonry	Joe | Jack	1
carpentry	Joe|Jim	1
roofing	Joe | Jim	1
plumbing	Jack	1
ceiling	Joe | Jim	1
windows	Joe | Jim	1
façade	Joe | Jack	1
garden	Joe | Jim | Jack	1
painting	Jack | Jim	1
movingIn	Joe | Jim	1

Execute Decision Model

Rule Solver will produces new results telling exactly “who does what”:

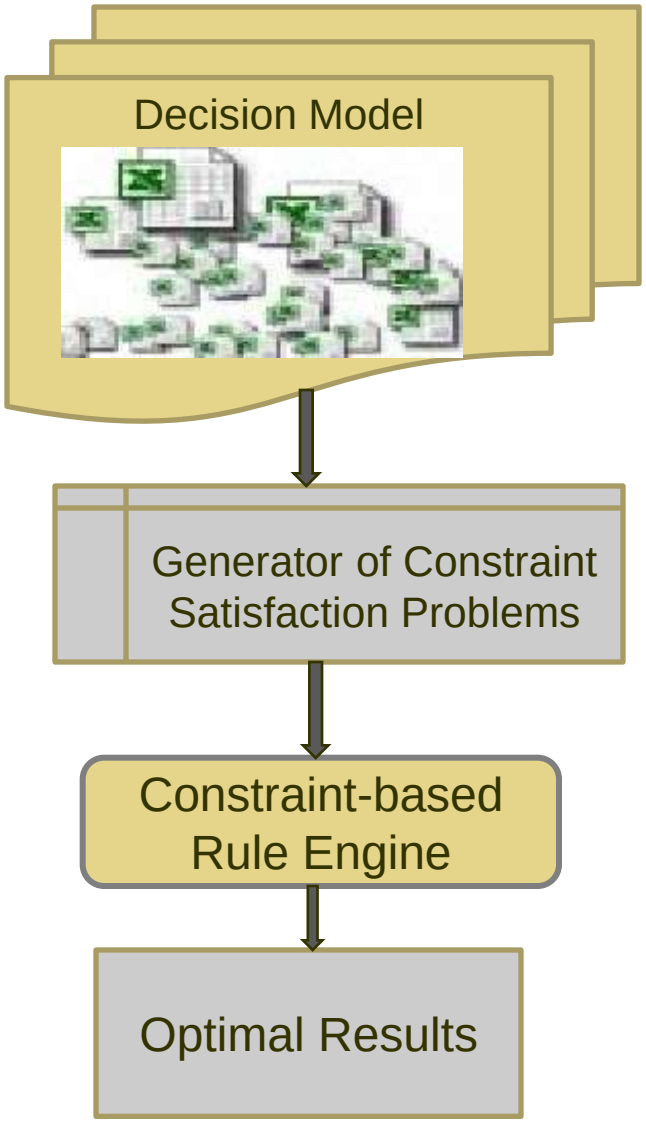
```
masonry[0 -- 7 --> 7)        requires Jack[1]
carpentry[7 -- 3 --> 10)     requires Jim[1]
roofing[10 -- 1 --> 11)      requires Jim[1]
plumbing[7 -- 8 --> 15)      requires Jack[1]
ceiling[7 -- 3 --> 10)       requires Joe[1]
windows[11 -- 1 --> 12)      requires Jim[1]
façade[15 -- 2 --> 17)       requires Jack[1]
garden[15 -- 1 --> 16)       requires Jim[1]
painting[0 -- 2 --> 2)       requires Jim[1]
movingIn[17 -- 1 --> 18)     requires Jim[1]
```

Benefits

- Executable Decision Models
- No rule languages to learn
  - no coding is required
- No proprietary GUI for rules management
  - business analysts create and execute decision models using Excel or Google Docs
- No IT involvement in business logic
- Easy to use and to integrate with Java/.NET

How Rule Solver Works

- Rule Solver reads and validates an Excel-based decision model
- Rule Solver generates “on the fly” a constraint satisfaction problem using the standard Constraint Programming API “JSR-331”
- Rule Solver solves the problem using a JSR-331 compliant constraint solvers



Off-the-shelf Constraint Solvers

- CP solvers are very powerful tools that utilize the results of the last 25 years R&D done by CP experts around the globe
- Rule Solver is based on the JSR-331 standard that allows a user to switch between different underlying CP solvers without any changes in the decision model
- Modeling facilities of Rule Solver are simple, intuitive, extendable, and oriented to business analysts (as well as programmers)
- However, internal implementations of the resource requirement constraints and search strategies can be very complex and for large-scale problems special search strategies may be required

Conclusion

- Tabular decision models can be successfully used even for complex scheduling and resource allocation problems
- Concentrating on “WHAT” rather than on “HOW” is a natural approach for a modeling environment such as Rule Solver that is oriented to subject matter experts
- At the same time, Rule Solver provides a special Java API to do more complex modeling in Java or mix both approaches
- You may download and try open sourced Rule Solver from www.openrules.com