JSR 331
“Constraint Programming API”

CON3255

Jacob Feldman, PhD

JSR331 Specification/Maintenance Lead
OpenRules, Inc., CTO
jacobfeldman@openrules.com

October 1, 2012
Shock Troops for Business Innovation

“I have concluded that decision making and the techniques and technologies to support and automate it will be the next competitive battleground for organizations. Those who are using business rules, data mining, analytics and optimization today are the shock troops of this next wave of business innovation”

Tom Davenport
Optimization Engines

- **Optimization** usually refers to a mathematical technique used to calculate the best possible resource utilization to achieve a desired optimization objective such as:
  - minimizing expenses or travel time
  - maximizing ROI or service level, etc.

- **Optimization Engine**:
  - Determines how to most effectively allocate resources, automatically balancing trade-offs and business constraints
  - Eliminates the need to manually work out plans and schedules, so your customers can achieve maximum operational efficiency

- **Leading Optimization Techniques**
  - LP/MIP – Linear and Mixed Integer Programming
  - CP – Constraint Programming
Constraint Programming (CP)

- Constraint Programming (CP) is a proven optimization technology supported by many off-the-shelf solvers with C++ and Java APIs.

- Constraint Programming has strong roots in Operation Research and AI:
  - ACP - Association for Constraint Programming

- CP is especially successful dealing with real-world scheduling, resource allocation, and complex configuration problems:
  - CP clearly separates Problem Definition from Problem Resolution bringing declarative programming to the real-world
  - CP makes different optimization techniques handily available to regular software developers (without PhD in Operation Research)
Typical CP Applications

- Scheduling and Resource Allocation
- Complex Configuration Problems
- Supply Chain Management
- Staff Rostering
- Vehicle Routing
Some Popular CP Tools

- Java API
  - Choco, Constrainer, JaCoP, JSetL, Cream

- C++ API
  - IBM/ILOG CP – Commercial (www.ilog.com)
  - Gecode – Open Source (www.gecode.org)

- CP environments with specialized modeling languages
  - OPL from IBM ILOG (www.ilog.com)
  - MiniZinc from G12 group, Australia (http://www.g12.cs.mu.oz.au)
  - Comet, Brown University (www.comet-online.org)
  - Prolog-based tools (ECLiPSe, SICStus)

- 20+ other CP Solvers: http://slash.math.unipd.it,cp/

- CP Solvers are usually well integrated with other optimization tools (LP, MIP)

- Lack of Standardization
JSR 331: Constraint Programming API

- JSR 331 became an official JCP standard in March 2012 and is in a maintenance mode
- This session describes:
  - Major Constraint Programming concepts
  - Practical examples of constraint satisfaction and optimization problems including scheduling and resource allocation
  - How OpenRules uses JSR 331 as an optimization component of its open source decision management system
JSR 331: Key Objectives

- Make CP more accessible for business software developers
- Allow a Java business application developer to easily switch between different CP solver implementations _without any changes_ in the application code
- Assist CP vendors in creating practical and efficient JSR 331 implementations
Procedural vs Declarative Programming

Simple example of a constraint satisfaction problem:

There are three integers $x$, $y$, and $z$ defined from 0 to 10. Our goal is to find the solution that would maximize or minimize the objective function represented by the following integer expression:

$cost = 3x + 4y - 4z$

subjected to:

$x < y$
$x + y = z$

A Pure Java solution

A CP-based solution
Java solution

```java
public static void main(String[] unused) {
    for(int x = 0; x <= 10; x++) {
        for (int y = 0; y <= 10; y++) {
            for (int z = 0; z <= 10; z++) {
                if (x < y && (x + y == z))
                    System.out.println("x="+x + " y="+y + " z=" +z);
            }
        }
    }
}
```

```java
public static void main(String[] unused) {
    for(int x = 0; x < 10; x++) {
        for (int y = x + 1; y <= 10; y++) {
            int z = x + y;
            if (z <= 10)
                System.out.println("x="+x + " y="+y + " z=" +z);
        }
    }
}
```
public class Test {

    public static void main(String[] args) {
        // ==== PROBLEM DEFINITION ==================================
        Problem p = ProblemFactory.newProblem("Test");
        // ====== Define variables
        Var x = p.variable("X", 1, 10);
        Var y = p.variable("Y", 1, 10);
        Var z = p.variable("Z", 1, 10);
        Var cost = x.multiply(3).multiply(y).minus(z.multiply(4));
        // ====== Define and post constraints
        p.post(x, "<", y);  // X < Y
        p.post(x.plus(y), "=" , z);  // X + Y = Z

        // === PROBLEM RESOLUTION ==================================
        p.log("=== Find Solution:");
        Solver solver = p.getSolver();
        Solution solution = solver.findSolution();
        if (solution != null)
            solution.log();
        else
            p.log("No Solution");
        p.log("Cost " + cost);
    }
}
Constraint Satisfaction Problem - CSP

- Typical CSP structure:

1. Problem Definition (what to do)
   a. Define Constrained Variables with all possible values
   b. Define Constraints on the variables

2. Problem Resolution (how to do it)
   a. Find Solution(s) that defines a value for each variable such that all constraints are satisfied or
   b. Find an optimal solution that minimizes/maximizes a certain optimization objective
How the constraint “$X < Y$” works

- Let’s assume $X$ and $Y$ are defined on the domain $[0,10]$
- Initial constraint propagation after posting $X < Y$ constraint:
  $X[0;9]$
  $Y[1;10]$
- Changes in $X$ cause the changes in $Y$
  $X > 3 \rightarrow Y > 4$
- Changes in $Y$ cause the changes in $X$
  $Y \leq 8 \rightarrow X \leq 7$
- Bi-Directional constraint propagation
Constraint Propagation (intuitive view)

User Actions: ”Small” Engines

Automatic Actions - “Big Engines”:
“Scheduler”, “Configurator”, “Router”, ...
JSR-331 CP API: Basic Concepts

- **Problem** – the main class that defines constraint satisfaction problems. A placeholder for all other objects and methods
  - **Var** – defines constraint integer variables
  - **VarReal** – defines constraint real variables
  - **VarBool** – defines constraint real variables
  - **VarSet** – defines constraint set variables
  - **Constraint** – defines various constraints

- **Solver** – the main class that solves constraint the problem
  - **Search Strategy**
  - **Solution**
CP API: Examples

• Analyse basic JSR-331 arithmetic examples:
  ◦ Test1 – find a solution
  ◦ Test2 – find all solutions
  ◦ Test3 – find an optimal solution

• Let’s analyse and run these problem in Eclipse IDE
CSP Example: “Map Coloring”

- A map-coloring problem involves choosing colors for the countries on a map in such a way that at most 4 colors are used and no two neighboring countries have the same color.
- We will consider six countries: Belgium, Denmark, France, Germany, Netherlands, and Luxembourg.
- The colors:
  - blue, white, red or green
Example “Map Coloring”: problem variables

Problem  p = ProblemFactory.newProblem("MapColoring");

// Define Variables
Var Belgium = p.variable("Belgium“,0, 3);
Var Denmark = p.variable("Denmark“,0, 3);
Var France = p.variable("France“,0, 3);
Var Germany = p.variable("Germany“,0, 3);
Var Netherlands = p.variable("Netherlands“,0, 3);
Var Luxemburg = p.variable("Luxemburg“,0, 3);

Each country is represented as a variable that corresponds to an unknown color: 0, 1, 2, or 3
“Map Coloring”: problem constraints

```java
p.post(France, "!=", Belgium);
p.post(France, "!=", Luxemburg);
p.post(France, "!=", Germany);
p.post(Luxemburg, "!=", Germany);
p.post(Luxemburg, "!=", Belgium);
p.post(Belgium, "!=", Netherlands);
p.post(Belgium, "!=", Germany);
p.post(Germany, "!=", Netherlands);
p.post(Germany, "!=", Denmark);
```
“Map Coloring”: solution search

Solution solution = p.getSolver().findSolution();
if (solution != null) {
    solution.log();
    for (int i = 0; i < p.getVars().length; i++) {
        Var var = p.getVars()[i];
        p.log(var.getName() + " - " + colors[solution getValue(var.getName())]);
    }
} else
    p.log("no solution found");

// Solution:
Belgium – red
Denmark – red
France – green
Germany – blue
Netherlands – green
Luxemburg - yellow
Solve Logical Puzzle

SEND + MORE = MONEY

This example shows how to represent and solve a simple puzzle:

\[
\begin{align*}
\text{SEND} \\
+ \text{MORE} \\
\hline
\text{MONEY}
\end{align*}
\]

where different letters represent different digits.

Eclipse: SendMoreMoney.java
Logical problem: “Zebra”

Here are the facts:
(1) There are 5 houses, each painted with a different color. The house is numbered from 1 to 5, from left to right.
(2) Each house is occupied by only one person; each person has a different nationality from the others.
(3) Each person drinks a different drink, smokes a different cigarette, and has a different pet.

Now, here are the known constraints:
(1) The British lives in the red house.
(2) The Spaniard has a dog.
(3) The owner of the green house drinks coffee.
(4) The Ukrainian drinks tea.
(5) The green house is located at the right side of the white house.
(6) The person who smokes OldGolds has a snail.
(7) The owner of the yellow house smokes Kools.
(8) The person who lives in the house exactly in the middle, drinks milk.
(9) The Norwegian lives in the house numbered one.
(10) The person who smokes Chesterfield lives next to the person who has a fox.
(11) The person who has a horse lives next to the person who smokes Kools.
(12) The person who smokes LuckyStrike drinks juice.
(14) The Norwegian lives next to the the blue house.

The question is:
Who has a ZEBRA?
Scheduling Problems

A special Java package: \texttt{org.jcp.jsr331.scheduler}
House Construction Problem

Eclipse: ScheduleActivities.java
Resource Allocation Problem

The following problem deals with activities that require a common resource. Let’s consider 5 different orders (activities) that fire batches of bricks in an oven (a resource with a limited capacity). Each order’s size and duration, as well as the oven’s capacity, are described in the following figure:

Eclipse: Ovens.java
Oven.java

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]
OpenRules “Rule Solver”

- **Rule Solver** provides an Excel-based interface for JSR-331
- Oriented to business analysts (non-programmers)
- Allows them to model and solve business decision optimization problems without learning Java
Example “Staff Rostering”

As the manager, you are required to hire and set a weekly schedule for your employees as follows:

- **Total employees required**

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>16</td>
<td>18</td>
<td>12</td>
</tr>
</tbody>
</table>

- **Available employees:**

<table>
<thead>
<tr>
<th>Employee Type</th>
<th>Total</th>
<th>Cost per Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>F/T</td>
<td>14</td>
<td>$100</td>
</tr>
<tr>
<td>P/T</td>
<td>4</td>
<td>$150</td>
</tr>
</tbody>
</table>

What is the minimal staffing cost?
Decision “DefineEmployeeSchedule”

- Presented in Excel file “Decision.xls”
- Decision Variables are defined in the glossary table:

<table>
<thead>
<tr>
<th>Decision Variable</th>
<th>Business Concept</th>
<th>Attribute</th>
<th>Domain</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon FT</td>
<td></td>
<td>monFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Mon PT</td>
<td></td>
<td>monPT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Tue FT</td>
<td></td>
<td>tueFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Tue PT</td>
<td></td>
<td>tuePT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Wed FT</td>
<td></td>
<td>wedFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Wed PT</td>
<td></td>
<td>wedPT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Thu FT</td>
<td></td>
<td>thuFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Thu PT</td>
<td></td>
<td>thuPT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Fri FT</td>
<td></td>
<td>friFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Fri PT</td>
<td></td>
<td>friPT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Sat FT</td>
<td></td>
<td>satFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Sat PT</td>
<td></td>
<td>satPT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Sun FT</td>
<td></td>
<td>sunFT</td>
<td>0-14</td>
<td>TRUE</td>
</tr>
<tr>
<td>Sun PT</td>
<td></td>
<td>sunPT</td>
<td>0-4</td>
<td>TRUE</td>
</tr>
<tr>
<td>Total Cost</td>
<td></td>
<td>totalCost</td>
<td>0-20000</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Decision “DefineEmployeeSchedule”

**Decision DefineEmployeeSchedule**

<table>
<thead>
<tr>
<th>Decisions</th>
<th>Execute Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Employee Daily Demand</td>
<td>:= EmployeeDailyDemand()</td>
</tr>
<tr>
<td>Define Total Cost</td>
<td>:= DefineTotalCost()</td>
</tr>
</tbody>
</table>

**Decision Table EmployeeDailyDemand**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Arith Oper</th>
<th>Variable</th>
<th>Compare Oper</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon FT</td>
<td>+</td>
<td>Mon PT</td>
<td>=</td>
<td>5</td>
</tr>
<tr>
<td>Tue FT</td>
<td>+</td>
<td>Tue PT</td>
<td>=</td>
<td>8</td>
</tr>
<tr>
<td>Wed FT</td>
<td>+</td>
<td>Wed PT</td>
<td>=</td>
<td>9</td>
</tr>
<tr>
<td>Thu FT</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fri FT</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sat FT</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sun FT</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Decision Table DefineTotalCost**

<table>
<thead>
<tr>
<th>Action</th>
<th>Scal Prod</th>
<th>Name of the Scalar Product</th>
<th>Numbers</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scal</td>
<td>Prod</td>
<td>Total Cost</td>
<td>100,150,150,100,150,100,150,100,150,100,150,100,150</td>
<td>Mon FT, Mon PT, Tue FT, Tue PT, Wed FT, Wed PT, Thu FT, Thu PT, Fri FT, Fri PT, Sat FT, Sat PT, Sun FT, Sun PT</td>
</tr>
<tr>
<td>Scal</td>
<td>Prod</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Run Decision from Java

```java
import com.openrules.ruleengine.Decision;

public class Main {
    public static void main(String[] args) {
        String fileName = "file:rules/Decision.xls";
        System.setProperty("OPENRULES_MODE", "Solve");
        Decision decision = new Decision("DefineEmployeeSchedule", fileName);
        decision.put("MaxSolutions", "30");
        decision.put("Minimize", "Total Cost");
        decision.execute();
        decision.execute("PrintSolution");
    }
}
```

==== Optimal Solution ====

<table>
<thead>
<tr>
<th>M</th>
<th>T</th>
<th>W</th>
<th>T</th>
<th>F</th>
<th>S</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>PT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Total Cost: 8100

====================
JSR-331 Implementations

Constraint Solvers
- Choco
- Constrainer
- JSetL
- JaCoP
- Cream
- ILOG CP
- Oscar
- Other

Common Constraint Solver Implementation

Common Linear Solver Implementation

Problem Description (JSR-331 Standard API)

Generated Standard Problem Representation (MPS/LP Format)

Off-the-shelf Constraint Solver (jar-files)

Off-the-shelf Linear Solver (External process)

Problem Solutions (JSR-331 Standard API)

Linear Solvers
- CPLEX
- GUROBI
- SCIP
- GLPK
- COIN
- LP-SOLVE
- OJALGO
- Other

Groovy and Scala APIs
Use Case “Cloud Balancing”

- You have a number of cloud computers and need to run a number of processes on those computers.
- Each process requires certain CPU power, RAM, and network bandwidth and incurs a certain maintenance cost (which is fixed per computer).
- Objective: assign process to computers while minimize the total maintenance cost.
Variable Computer

- Input classes: CloudComputer, CloudProcess
- Decision variables are in this class:

```java
public class VarComputer {
    CloudComputer computer;
    Var[] processVars; // processVars[i] = 1 means this computer is used by the i-th process

    public VarComputer(Problem p, CloudComputer computer, CloudProcess[] processes,
                       int[] requiredMemories, int[] requiredCpuPowers, int[] requiredNetworkBandwidths) {
        this.computer = computer;
        processVars = new Var[processes.length];
        for (int i = 0; i < processes.length; i++) {
            String name = "P" + processes[i].getId() + "C" + computer.getId();
            processVars[i] = p.variable(name, 0, 1);
        }
        p.post(requiredMemories, processVars, "\leq", computer.getMemory());
        p.post(requiredCpuPowers, processVars, "\leq", computer.getCpuPower());
        p.post(requiredNetworkBandwidths, processVars, "\leq", computer.getNetworkBandwidth());
    }

    public Var[] getProcessVars() {
        return processVars;
    }
}
```
Modeling and Search for an Optimal Solution

- **A small problem “4 x 12”** with a constraint solver
  - 4 computers and 12 processes
  - “Brute force” approach: 650mills
  - “Sort processes first” approach: 490 mills

- **A larger problem “10 x 20”**
  - **Constraint solver** takes 30 seconds
    - (50x longer) and only when we set a time limit
  - **Linear Solver** (identical source code, just different jars in classpath)
    - Finds an optimal solution in 1,200 milliseconds
Modeling and Search for an Optimal Solution (2)

- **A large problem “50 x 100”**
  - 50 computers and 100 processes
  - Constraint solver requires special selectors and time limits
  - Linear Solver takes 2.5 hours to find an optimal solution

- **A huge problem “5,000 x 55,000”**
  - Offered at the recent [ROADEF-2012](https://www.roadef.com) competition
  - The winners found the best solution within 5 mins using a unique selection of subsets of processes and computers and a specially written solver
JSR 331 Free Downloads

- Open Source
- Download JSR-331
  - http://openrules.com/jsr331
- Detailed Documentation
- Want to contribute?
  - Email to jacobfeldman@openrules.com to get an SVN access
- Rule Solver
  - http://openrules.com/rulesolver.htm