The 19th International Conference on the Principles and Practice of Constraint Programming

The workshop "CP Solvers-2013: Modeling, Applications, Integration, and Standardization"

JSR331 Standard:
Current State and Future Plans

www.jsr331.org

Jacob Feldman

JSR331
Specification and Maintenance Lead

Uppsala, Sweden
September 16, 2013

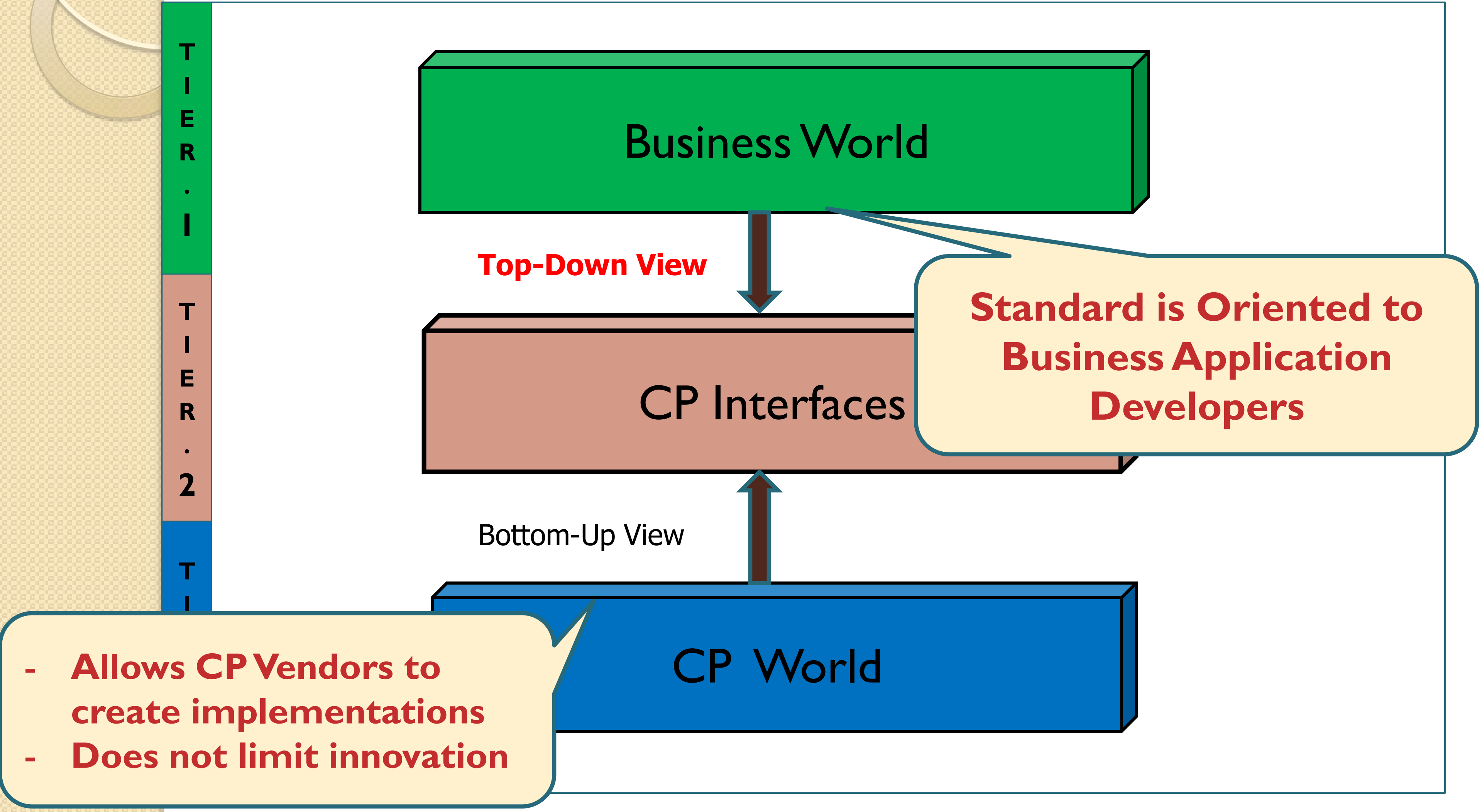
JSR331 – Java CP API Standard

- The standard for Java Constraint Programming API has been developed under the terms of the Java Community Process (JCP) www.jcp.org
- JSR331 covers major CP concepts for representation and resolution of constraint satisfaction and optimization problems
- JSR331 is open sourced, free, and comes with working implementations, detailed documentation, and multiple examples

Key Objectives of the Standard

- Make CP more accessible for business application developers
- Allow an easy switch between different solver implementations without any changes in the application code
- Assist CP vendors in creating practical and efficient JSR331 implementations

CP Standardization Approach



TIER.1

TIER.2

TI

Business World

Top-Down View

CP Interfaces

Bottom-Up View

CP World

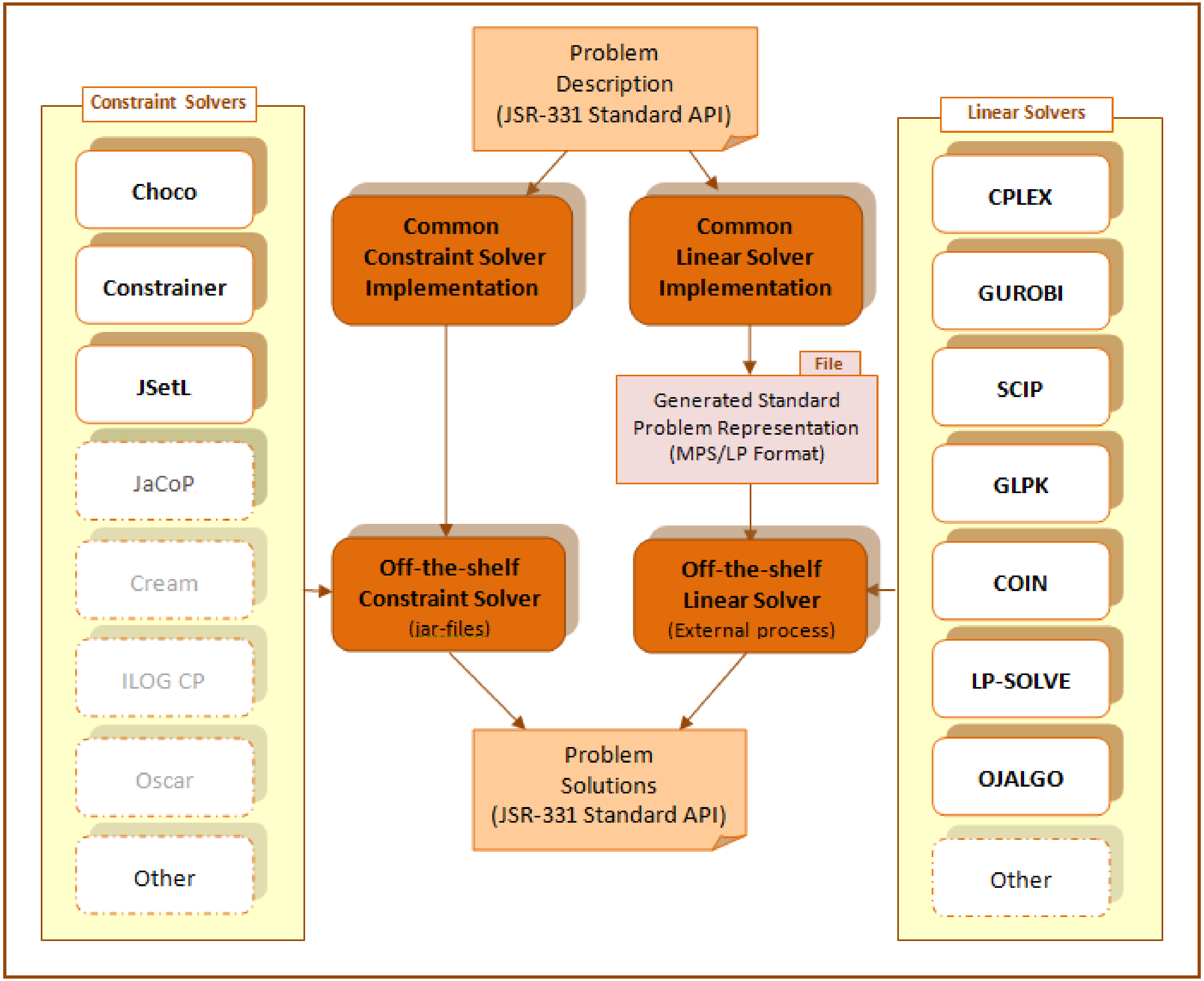
Standard is Oriented to Business Application Developers

- Allows CP Vendors to create implementations
- Does not limit innovation

Key Milestones

- Aug-2009:
  - Initiated at JCP, supported by ACP
  - Expert group and CP community involvement, CP2009,2010
- Sep-2010:
  - 3 initial implementations, manuals, free downloads and community testing
  - JavaOne-2010 Award "The Most Innovative JSR"
- Mar-2012:
  - Approved by JCP Executive Committee as an official standard
- Oct-2012:
  - New Maintenance Release with 3 CP and 7 LP implementations
- A new release - by the end of 2013

Current JSR331 Implementations



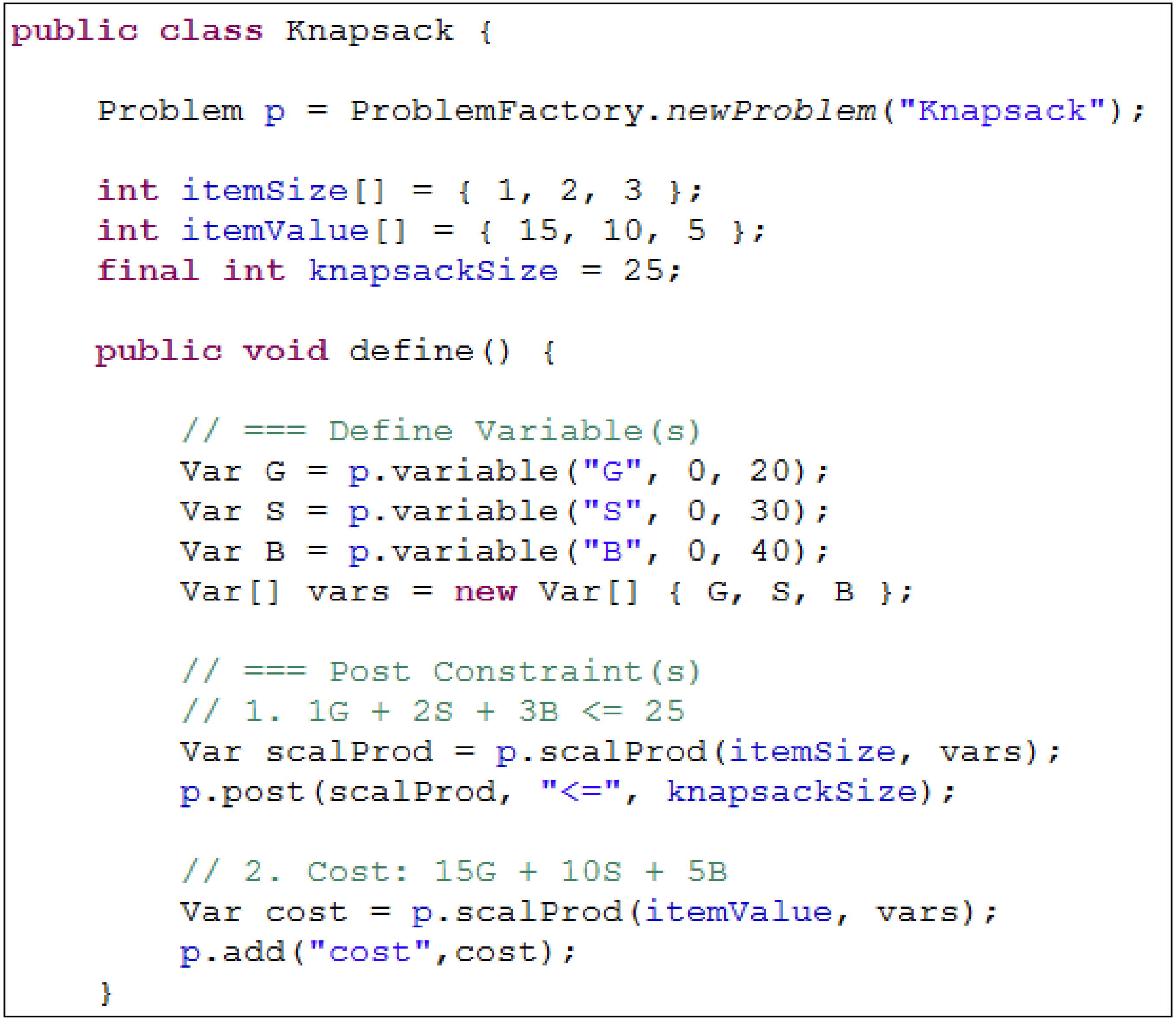
Problem Definition Concepts

- Problem
- Constrained Variables
    - Var
    - VarReal
    - VarBool
    - VarSet
    - VarString
- Constraints
    - All Arithmetic
    - All Logical
    - Major Global: Linear, AllDiff, Element, Cardinality, GCC
    - Custom

Problem Resolution

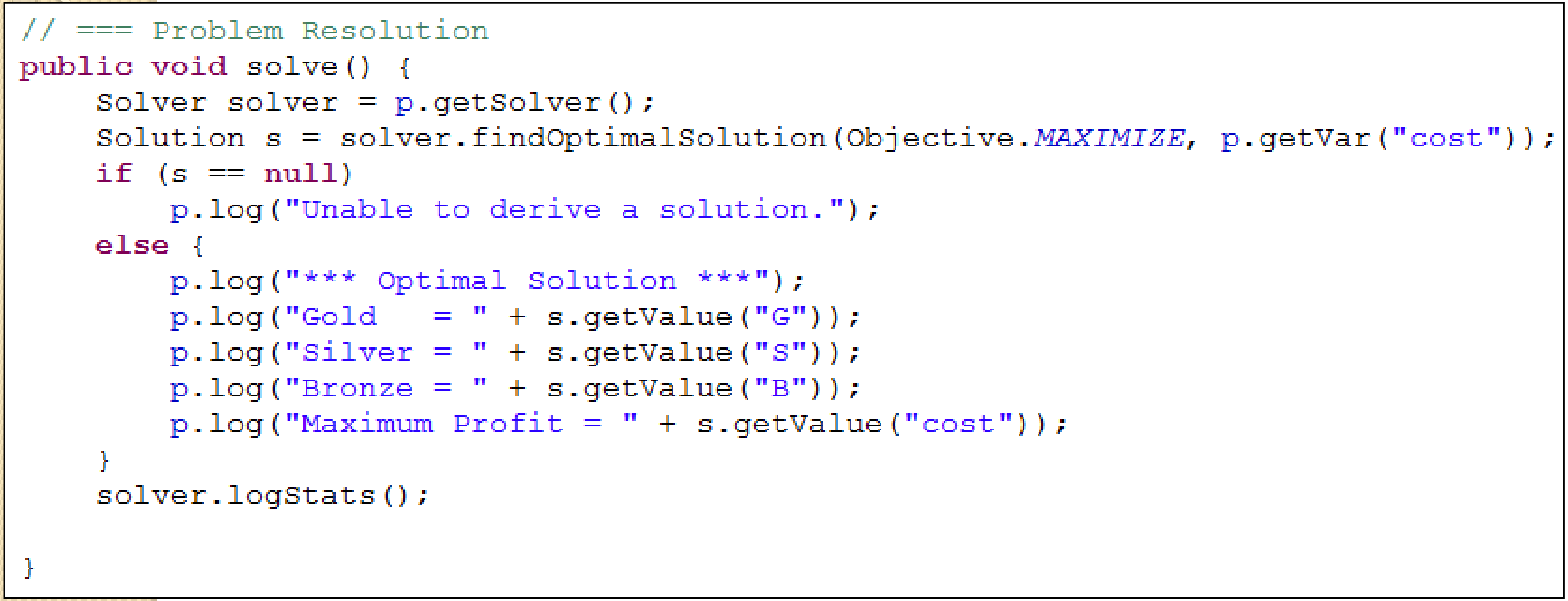
- Solver – a factory and a placeholder for search related objects
- Search Methods
  - Find One Solution
  - Find Optimal Solution
  - Iterate through solutions within user-defined limits
- SearchStrategy
  - The default strategy and implementation strategies
  - Variable and Value Selectors
- Solution

Knapsack: Source Code with JSR331



```
public class Knapsack {

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

    int itemSize[] = { 1, 2, 3 };
    int itemValue[] = { 15, 10, 5 };
    final int knapsackSize = 25;

    public void define() {

        // === Define Variable(s)
        Var G = p.variable("G", 0, 20);
        Var S = p.variable("S", 0, 30);
        Var B = p.variable("B", 0, 40);
        Var[] vars = new Var[] { G, S, B };

        // === Post Constraint(s)
        // 1. 1G + 2S + 3B <= 25
        Var scalProd = p.scalProd(itemSize, vars);
        p.post(scalProd, "<=", knapsackSize);

        // 2. Cost: 15G + 10S + 5B
        Var cost = p.scalProd(itemValue, vars);
        p.add("cost",cost);
    }
```

Knapsack: Source Code with JSR331



```java
// === Problem Resolution
public void solve() {
    Solver solver = p.getSolver();
    Solution s = solver.findOptimalSolution(Objective.MAXIMIZE, p.getVar("cost"));
    if (s == null)
        p.log("Unable to derive a solution.");
    else {
        p.log("*** Optimal Solution ***");
        p.log("Gold   = " + s.getValue("G"));
        p.log("Silver = " + s.getValue("S"));
        p.log("Bronze = " + s.getValue("B"));
        p.log("Maximum Profit = " + s.getValue("cost"));
    }
    solver.logStats();

}
```

Use CP Solver (e.g. Choco)

- Classpath contains:
  - jsr331.choco.jar
  - choco-solver-2.1.5-20120603.jar

```
RESULTS:
JSR-331 Standard v.1.1.0 Beta (release 7/26/2012)
JSR-331 Implementation based on CHOCO 2.1.5, build
2012.06.03
*** Optimal Solution ***
Gold   = 20
Silver = 2
Bronze = 0
Maximum Profit = 320
Execution time: 515 msec
```

Use LP Solver (e.g. Gurobi)

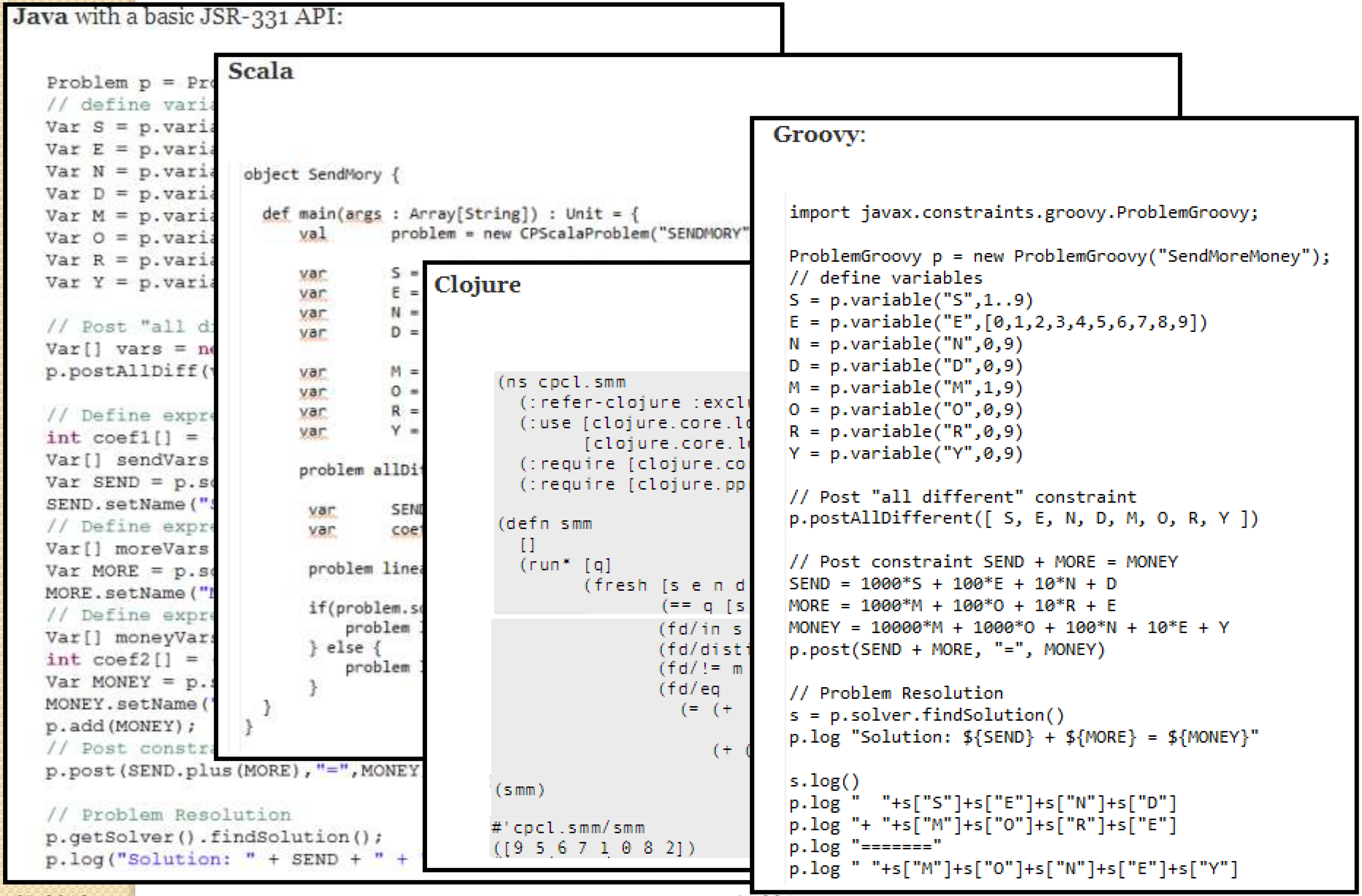
- The same source but Classpath contains:
  - jsr331.linear.jar
  - jsr331.gurobi.jar

```
RESULTS:
JSR-331 Standard v.1.1.0 Beta (release 7/26/2012)
JSR-331 Implementation based on Linear Solver
Solve problem using GUROBI v.5.0.1
Execute command: gurobi_cl Threads=1
ResultFile=results/Knapsack.sol results/Knapsack.mps
*** Optimal Solution ***
Gold   = 20
Silver = 2
Bronze = 0
Maximum Profit = 320

Execution time: 1322 msec
```

TCK – Technology Compatibility Kit

- **org.jcp.jsr331.tests**
  - JUnit tests with asserts for expected results
  - Only these tests are mandatory for compliance
- **org.jcp.jsr331.samples**
  - A library of well-known CSPs implemented using a pure JSR-331 API
- **org.jcp.jsr331.hakan**
  - A library of problems created by Hakan Kjellerstrand

JSR331 in JVM Languages



Java with a basic JSR-331 API:

```
Problem p = Pro
// define varia
Var S = p.varia
Var E = p.varia
Var N = p.varia
Var D = p.varia
Var M = p.varia
Var O = p.varia
Var R = p.varia
Var Y = p.varia

// Post "all di
Var[] vars = ne
p.postAllDiff(v

// Define expre
int coef1[] =
Var[] sendVars
Var SEND = p.sc
SEND.setName("S
// Define expre
Var[] moreVars
Var MORE = p.sc
MORE.setName("M
// Define expre
Var[] moneyVars
int coef2[] =
Var MONEY = p.s
MONEY.setName("
p.add(MONEY);
// Post constra
p.post(SEND.plus(MORE),"=",MONEY

// Problem Resolution
p.getSolver().findSolution();
p.log("Solution: " + SEND + " +
```

Scala

```
object SendMory {

  def main(args : Array[String]) : Unit = {
      val        problem = new CPScalaProblem("SENDMORY"

      var        S =
      var        E =
      var        N =
      var        D =

      var        M =
      var        O =
      var        R =
      var        Y =

      problem allDi

       var       SEND
       var       coef

       problem linea

       if(problem.so
          problem
       } else {
          problem
       }
  }
}
```

Clojure

```
(ns cpcl.smm
  (:refer-clojure :excl
  (:use [clojure.core.lo
        [clojure.core.lo
  (:require [clojure.co
  (:require [clojure.pp

(defn smm
  []
  (run* [q]
        (fresh [s e n d
               (== q [s
               (fd/in s
               (fd/disti
               (fd/!= m
               (fd/eq
                 (= (+

                    (+ (

(smm)

#'cpcl.smm/smm
([9 5 6 7 1 0 8 2])
```

Groovy:

```
import javax.constraints.groovy.ProblemGroovy;

ProblemGroovy p = new ProblemGroovy("SendMoreMoney");
// define variables
S = p.variable("S",1..9)
E = p.variable("E",[0,1,2,3,4,5,6,7,8,9])
N = p.variable("N",0,9)
D = p.variable("D",0,9)
M = p.variable("M",1,9)
O = p.variable("O",0,9)
R = p.variable("R",0,9)
Y = p.variable("Y",0,9)

// Post "all different" constraint
p.postAllDifferent([ S, E, N, D, M, O, R, Y ])

// Post constraint SEND + MORE = MONEY
SEND = 1000*S + 100*E + 10*N + D
MORE = 1000*M + 100*O + 10*R + E
MONEY = 10000*M + 1000*O + 100*N + 10*E + Y
p.post(SEND + MORE, "=", MONEY)

// Problem Resolution
s = p.solver.findSolution()
p.log "Solution: ${SEND} + ${MORE} = ${MONEY}"

s.log()
p.log "   "+s["S"]+s["E"]+s["N"]+s["D"]
p.log "+ "+s["M"]+s["O"]+s["R"]+s["E"]
p.log "======="
p.log "  "+s["M"]+s["O"]+s["N"]+s["E"]+s["Y"]
```

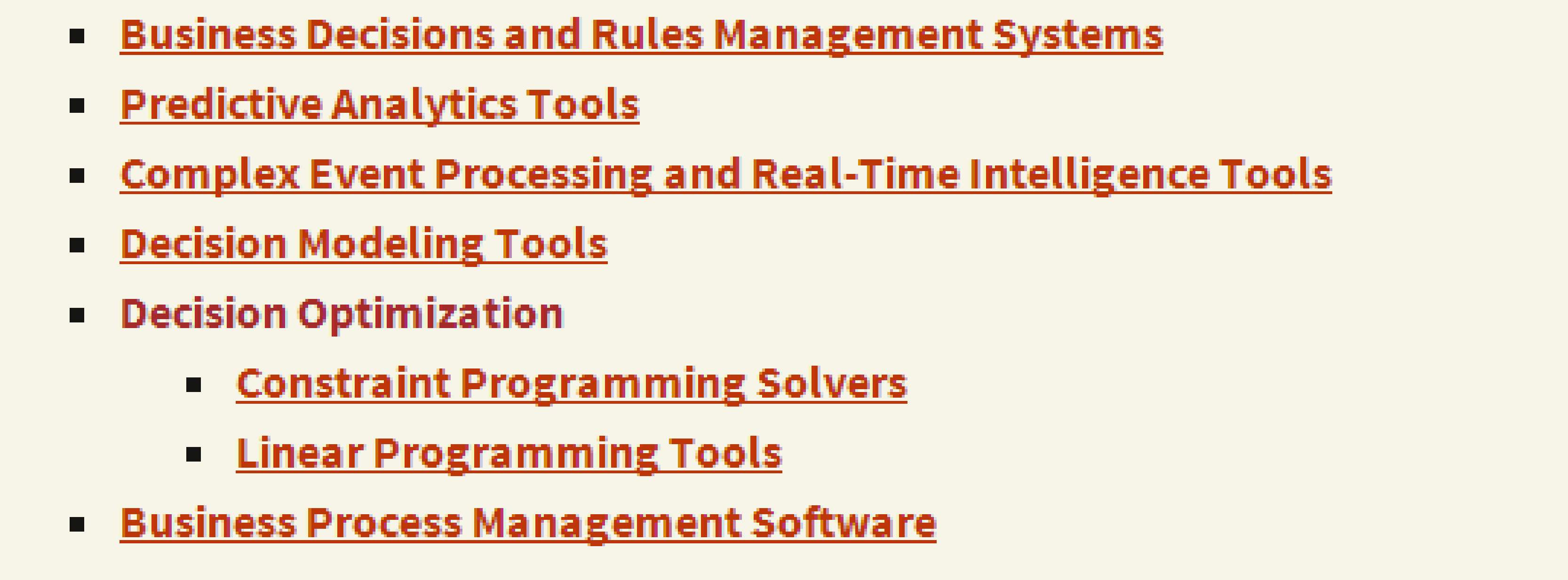
Business Decision Optimization

- Decision Optimization becomes an important component of any modern Business Decision Management System (BDMS)
  - See for instanse IBM ODM "golden topology"
- BDMS is usually oriented to Business Analysts
- Optimization brings a new power to BDMS beyond traditional rule engines
- A combination of CP/LP/MIP tools provide a solid foundation and JSR331 provides a standardized interface
- Decision Tools Catalogues decision-tools.org

www.decision-tools.org

Decision Management Tools

Software Tools for Decision Management.......................www.decision-tools.org



- Business Decisions and Rules Management Systems
- Predictive Analytics Tools
- Complex Event Processing and Real-Time Intelligence Tools
- Decision Modeling Tools
- Decision Optimization
  - Constraint Programming Solvers
  - Linear Programming Tools
- Business Process Management Software

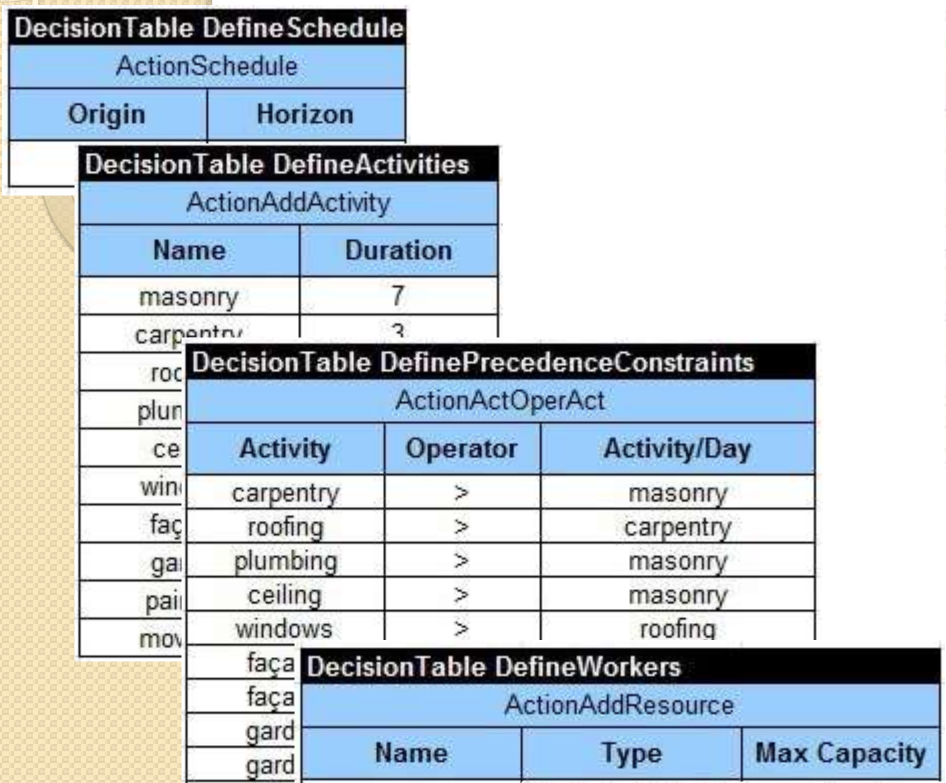
BDMS Integration Example

- OpenRules®
  - A popular Open Source Business Rules and Decision Management System
- Includes an inferential rule engine implemented using JSR331-compliant solvers
- OpenRules Rule Solver also supports:
  - Scheduling and Resource Allocation
  - Rules Conflict Diagnostics and Resolution

Scheduling Example

- OpenRules Rule Solver uses the basic JSR331 scheduler written in a solver independent way
- Includes Excel-based decision tables for "Scheduling and Resource Allocation Problems"
- Provides examples of various house construction decisions

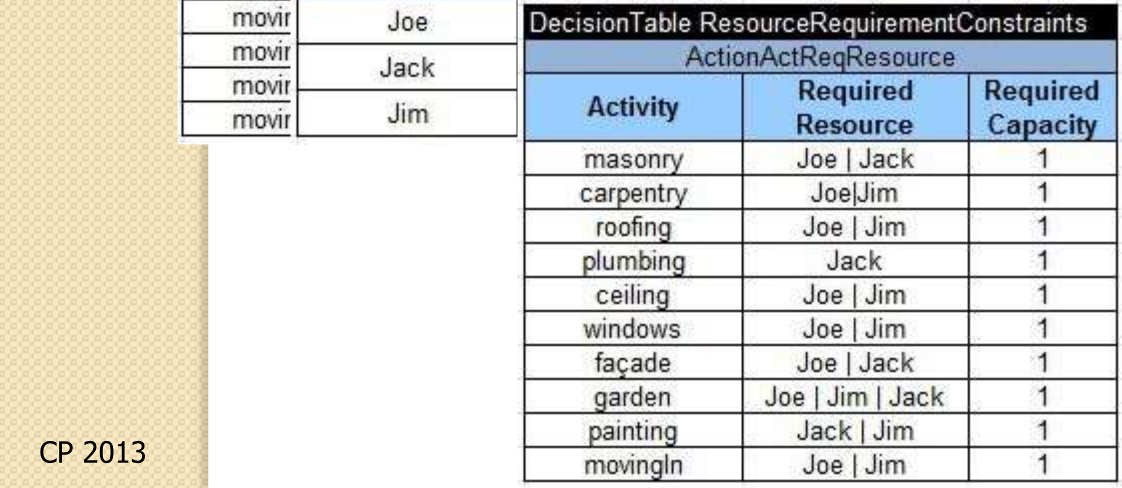
Decision: Schedule With Alternative Resources



DecisionTable DefineSchedule	
ActionSchedule	
Origin	Horizon

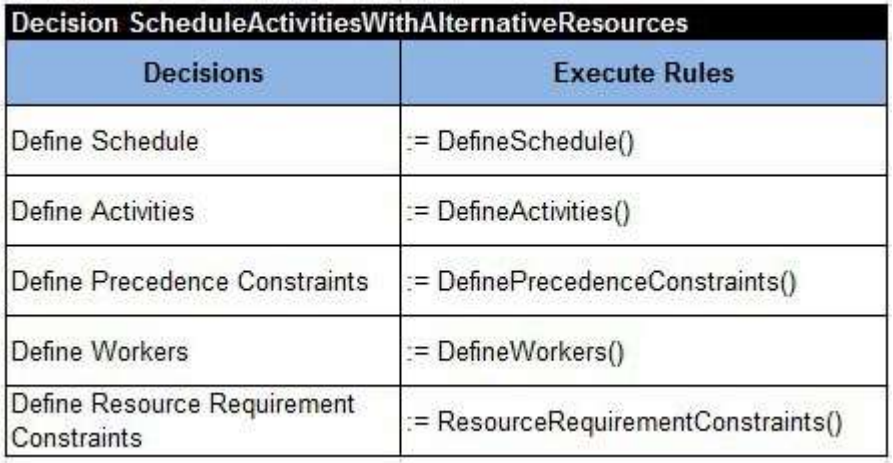
DecisionTable DefineActivities	
ActionAddActivity	
Name	Duration
masonry	7
carpentry	3
roo	
plun	
ce	
win	
faç	
ga	
pai	
mov	

DecisionTable DefinePrecedenceConstraints		
ActionActOperAct		
Activity	Operator	Activity/Day
carpentry	>	masonry
roofing	>	carpentry
plumbing	>	masonry
ceiling	>	masonry
windows	>	roofing
faça		
faça		
gard		
gard		
movir		
movir		
movir		
movir		

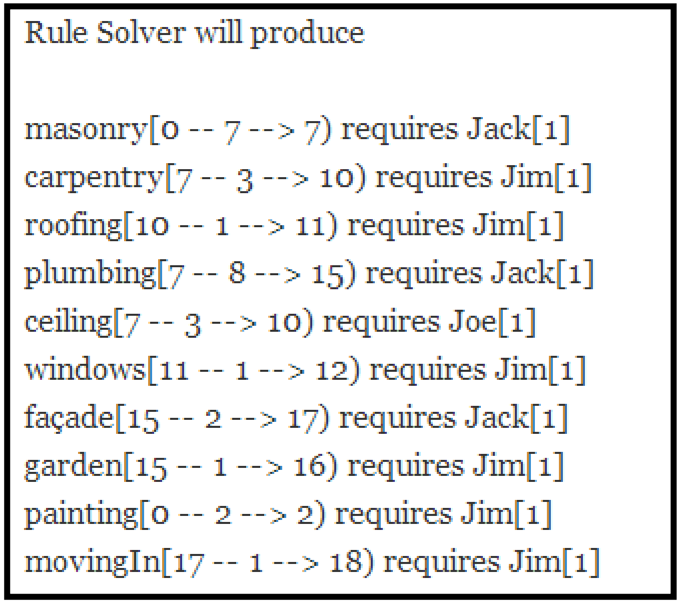


DecisionTable DefineWorkers		
ActionAddResource		
Name	Type	Max Capacity
Joe		
Jack		
Jim		

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



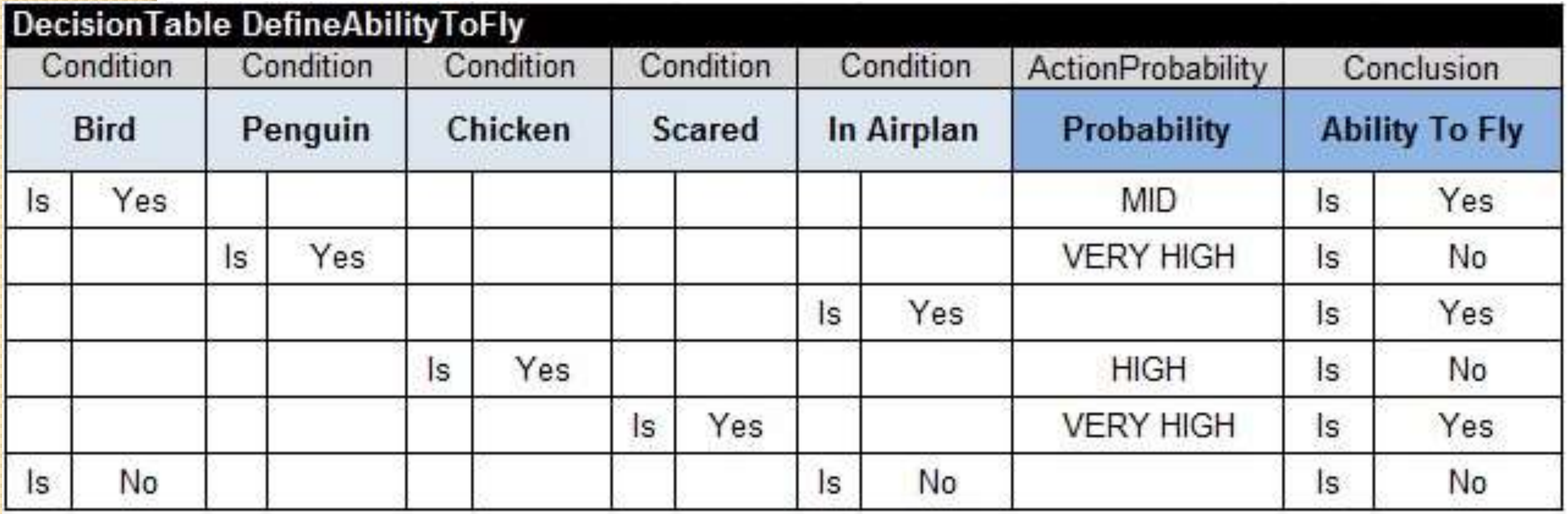
Decision ScheduleActivitiesWithAlternativeResources	
Decisions	Execute Rules
Define Schedule	:= DefineSchedule()
Define Activities	:= DefineActivities()
Define Precedence Constraints	:= DefinePrecedenceConstraints()
Define Workers	:= DefineWorkers()
Define Resource Requirement Constraints	:= ResourceRequirementConstraints()



```
Rule Solver will produce

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]
```

Resolving Rule Conflicts

- CP-based Implementation of the Defeasible Logic
- Example:
  - Rule 1: Birds can fly
  - Rule 2: Penguins do not fly
  - Rule 3: Chickens do not fly
  - Rule 4: Scared chickens do fly
  - Rule 5: Everybody can fly in the airplane



DecisionTable DefineAbilityToFly												
Condition		Condition		Condition		Condition		Condition		ActionProbability	Conclusion	
Bird		Penguin		Chicken		Scared		In Airplan		Probability	Ability To Fly	
Is	Yes									MID	Is	Yes
		Is	Yes							VERY HIGH	Is	No
								Is	Yes		Is	Yes
				Is	Yes					HIGH	Is	No
						Is	Yes			VERY HIGH	Is	Yes
Is	No							Is	No		Is	No

JSR331 Availability

- Download from www.jsr331.org
- Documentation
  - Specification
  - User Manual
  - Java Doc
- Working software with 3 implementations:
  - Choco
  - Constrainer
  - JSetL
- Everything is free and open sourced

JSR331 Future Plans

- Complete support for Real Constrained Variables
- Additional Global Constraints
- MiniZinc Integration
- More solver implementations
  - Cream, OR-tools, OscaR, …
- Verticals
  - Advance JSR331 Scheduler
  - Develop JSR331 Router (any volunteer?)
  - Develop JSR331 Configurator (any volunteer?)